TURBO C®

REFERENCE GUIDE

BORLAND

TURBO C®

Borland's No-Nonsense License Statement!

This software is protected by both United States copyright law and international treaty provisions. Therefore, you must treat this software *just like a book*, with the following single exception. Borland International authorizes you to make archival copies of the software for the sole purpose of backing-up our software and protecting your investment from loss.

By saying, "just like a book," Borland means, for example, that this software may be used by any number of people and may be freely moved from one computer location to another, so long as there is **no possibility** of it being used at one location while it's being used at another. Just like a book that can't be read by two different people in two different places at the same time, neither can the software be used by two different people in two different places at the same time (unless, of course, Borland's copyright has been violated).

Programs that you write and compile using the Turbo C language compiler may be used, given away or sold without additional license or fees, as long as all copies of such programs bear a copyright notice. By "copyright notice" we mean either your own copyright notice or, if you prefer, the statement, "Created using Turbo C, Copyright © Borland 1987, 1988." Included in the Turbo C diskettes are several support files that contain encoded hardware and font information used by the standard graphics library (GRAPHICS.LIB). These files, which can be listed by typing DIR *.CHR and DIR *.BGI, are proprietary to Borland International. You may use these files with the programs you create with Turbo C for your own personal use. In addition, to the extent the programs you write and compile using the Turbo C language compiler make use of these support files, you may distribute these support files in combination with such programs, provided that you do not use, give away, or sell the support files separately, and all copies of such programs bear a copyright notice.

The sample programs included on the Turbo C diskettes provide a demonstration of how to use the various features of Turbo C. They are intended for educational purposes only. Borland International grants you (the registered owner of Turbo C) the right to edit or modify these sample programs for your own use, but you may not give away or sell them, alone or as part of any program, in executable, object or source code form. You may, however, incorporate miscellaneous sample program routines into your programs, as long as your resulting programs do not substantially duplicate all or part of a sample program in appearance or functionality and all copies of such programs bear a copyright notice.

Limited Warranty

With respect to the physical diskette and physical documentation enclosed herein, Borland International, Inc. ("Borland") warrants the same to be free of defects in materials and workmanship for a period of 60 days from the date of purchase. In the event of notification within the warranty period of defects in material or workmanship, Borland will replace the defective diskette or documentation. **If you need to return a product, call the Borland Customer Service Department to obtain a return authorization number.** The remedy for breach of this warranty shall be limited to replacement and shall not encompass any other damages, including but not limited to loss of profit, and special, incidental, consequential, or other similar claims.

Borland International, Inc. specifically **disclaims** all other warranties, expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose with respect to defects in the diskette and documentation, and the program license granted herein in particular, and without limiting operation of the program license with respect to any particular application, use, or purpose. **In no event shall Borland be liable for any loss of profit or any other commercial damage, including but not limited to special, incidental, consequential or other damages.**

Governing Law

This statement shall be construed, interpreted, and governed by the laws of the state of California. Use, duplication, or disclosure by the U.S. Government of the computer software and documentation in this package shall be subject to the restricted rights under DFARS 52.227-7013 applicable to commercial computer software.

専門プログラマのためのTurbo C 2.0 ランタイム・ライブラリ・ソースコード

Turbo C ランタイム・ライブラリ・ソースコードによって、皆さんは、最初から専門家として活躍できます。このソースコードにより、皆さんはTurbo C についての理解を深めることができ、また、コンパイラ・ルーチンおよびファンクション内部の仕組みについて詳しく知ることができますから、皆さんのプログラム環境のより良い管理が可能となります。

Turbo C ランタイム・ライブラリによって、300ファンクション以上のライブラリをカスタマイズすることができ、また、ルーチンを追加・変更できます。したがって、皆さんのプログラムの具体的ニーズが、Turbo C ランタイム・ライブラリと異なる場合にも対処することができます。また、厳密にコード化および最適化されたTurbo C ライブラリのルーチンを利用することができ、しかも、それが前提としている事柄に制限されることはありません。

コンパイラ会社には、ライブラリ・ソースコードを提供しないところもあり、その代金として数千ドルも請求する会社もあります。その結果、多くの専門家は、ライブラリ・ルーチンの動きについて十分理解するために、やむなく数千時間も費やして自分自身のライブラリを書かねばなりません。ボーランドではその手間を省いて、マイクロ・コンピュータの歴史において最も人気のあるコンパイラのソースコードを、わずか29,800円で皆さんにご利用いただこうというわけです。

今すぐTurbo C ランタイム・ライブラリ・ソースコードをご注文ください。注文書にご記入後「Turbo C ランタイム・ライブラリ・ソースコード注文」と表書きの上、下記までお送りください。
また、ご入金も下記の口座までお願いいたします。

代理店

株式会社マイクロソフトウェア　アソシエイツ（略称 MSA）
〒107　東京都港区南青山７－８－１小田急南青山ビル９Ｆ
TEL 03-486-1411　／　FAX 03-486-8905

お振込先：東海銀行　九段支店　当座７１０３５７

【注意】 裏面に必要事項をご記入後、切り取らずにこのままご返送ください。控えが必要な方はコピーをお取り下さい。

Turbo C 2.0 ランタイム・ライブラリ・ソースコード注文書

Turbo C シリアルナンバー　＿＿＿＿＿＿　**メディア・サイズ**　＿＿＿＿＿＿

▽個人で購入された方のみご記入ください。

お名前

ご住所　〒

TEL

▽会社で購入された方のみご記入ください。

お会社名

ご住所　〒

ご所属

ご担当（お名前）

TEL

お支払方法（該当するものに√をつけてください）

□銀行振込（振込日　　年　　月　　日：□文書　□電信）
□現金書留
□その他（　　　　　　）

※本契約書兼注文書の受領、および代金お支払の確認後１ヶ月以内に商品発送となります。

Turbo C 2.0 ランタイム・ライブラリ・ソースコード契約書

Turbo C ランタイム・ライブラリ・ソースコード実施契約についてボーランドはまたとない機会を提供します

ボーランドでは、今一度、絶好の機会を皆さんに提供いたします。経験の深いCプログラマとして、皆さんはTurbo C ランタイム・ライブラリ・ソースコードの取扱いがいかに重要であるかをご存知です。当社では、Turbo C ランタイム・ライブラリ・ソースコードを、たった29,800円で、日本のTurbo C ユーザーの皆さんに提供いたします。

これは、皆さんがコードをよりよく理解し、Turbo C プログラムの性能を最大限に活かすまたとない機会です。

今すぐTurbo C ランタイム・ライブラリ・ソースコードをご注文下さい。極めて貴重で信頼性の高いTurbo C ランタイム・ライブラリ・ソースコードを利用するために、どうぞこのページの注文書にご記入の上、代理店宛にお申し込み下さい。

Turbo C 2.0 ランタイム・ライブラリ・ソースコード／ボーランドの実用的な実施契約

ボーランド・インターナショナル（以下「ボーランド」）は、Turbo C の中に含まれる Turbo C ライブラリの部分のソースコード（以下「ソース・プログラム」）、および、別料金にて、後日ボーランドが提供する更新データ（ただし8087エミュレータとグラフィックス・ライブラリのソースコードは含まれません）の実施契約の募集を行っております。

このソース・プログラムは、米国著作権法および国際条約によって保護されておりますので、下記の唯一の例外事項を除いて書籍と同様の取扱いが必要です。皆さんは皆さんのプログラムのバックアップのため、それから皆さんの支出について損されないためにのみ、ソース・プログラムの保存用コピーを作成することができます。

「書籍と同様に」とは、たとえば、ソース・プログラムの使用者数に制限を設けず、複数のコンピュータ・ワークステーションで同時に使用される可能性がない限り、ソース・プログラムはその間を自由に移動することができるという意味です。書籍が同時に二ヶ所で二人の人間によって読まれるのが不可能であるのと同様に、ソース・プログラムも、同時に二ヶ所の異なった場所で二人の異なった人間によって使用することはできません。（もちろんボーランドの著作権を侵害すれば別です。）この制限付保証は、しかし、皆さん自身によるソース・プログラムの変更に起因した欠陥には適用されません。

ソース・プログラムは、Turbo C の実施許諾コピーをサポートするために使用できます。これはどういうことかと申しますと、ソース・プログラムを皆さん自身が開発する Turbo C ベースのプログラムのコピーに含めることができるということですが、それは実行可能な形式においてのみ、それらを配付できるという意味です。実際の Turbo C ランタイム・ライブラリ・ソースコードのいかなる部分も配付できません。もちろん、皆さん自身のソースコードの配付は制限されません。

皆さんはソース・プログラムを変更することはできますが、変更されたソースコードの所有権は、その変更の程度とは無関係に、ボーランドに属します。皆さんはボーランドの著作権、その他の財産権に関する表示を除去したり変更したりすることはできません。また、変更の程度とは無関係に、ソース・プログラムのいかなる部分も配付できず、他のコンピュータのオペレーティング・システムまたは環境に移すことはできません。皆さんは、皆さん自身によるソース・プログラムの変更、およびそのプログラムを含む製品に起因するすべての請求、責任および損害について責任を負わねばなりません。

この実施権によって明示的に付与されていない権利は、すべてボーランドが留保します。

制限付保証

実際のディスケットおよびドキュメンテーションについては、ボーランドはその材料およびその仕上に欠陥がないことを保証します。これは、購入後60日間の限定保証です。ボーランドが、この60日以内に、材料またはその仕上の欠陥について文書による通知を受けた場合には、そのディスケットまたはドキュメンテーションを交換いたします。皆さんが製品を返却したいと思う場合には、返却許可番号の決定手続きのために、代理店まで連絡してください。

また、ボーランドおよび代理店は、ソース・プログラムの使用に関し、技術的な援助は提供しません。

この制限付保証の違反に対する救済手段は、ディスケットおよびドキュメンテーションの代替に限定され、その他の損害賠償は含まれません。ボーランドは、たとえ当方の代理店がそのような損害の可能性についてあらかじめ通知されていても、利益の損失、またはその他の商業上の損害を含む間接損害、特別損害、その他の類似の損害または請求について責任を負いません。また、いかなる場合にも、皆さんまたはその他の者の損害に対するボーランドの責任は、請求の形式を問わず、ソース・プログラム実施権について支払われる代価を超えるものではありません。

ボーランドは、その他のいかなる明示または黙示の保証も行いません。具体的には、ボーランドは、ソース・プログラムが特定の用途に対して適合することを保証するものではありません。商業価値に対する保証は、60日の保証期間に限って実際のディスケットおよびドキュメンテーション（ソース・プログラムは含まれません）についてのみ行われる制限付保証で、その他については保証しないことを明示します。

この制限付保証により、皆さんは具体的な法的権利を付与されます。権利の内容は国は州により異なり、国や州によっては付随的または間接的損害の除去や黙示保証の期間制限を認めないところもあり、従って、前記の規定が皆さんに適用されない場合もあります。

準拠および一般規定

本実施契約は、カリフォルニア州法に基づき、解釈・規制されるものとし、そのいずれかの条項が無効または実施不可能の場合においても、本契約の他の条項の効力は影響を受けず、その条件通りに実行されるものとします。本契約に基づく救済手段が、その本来の目的を達成できない場合においても、本契約に定める責任の限定および損害賠償の除外に関するすべての規定は、全面的にその効力を持続するものとします。本契約は、皆さんおよびボーランドの権限ある代表者が署名した文書による以外、変更できないものとします。

私は、前記について読了し、了解いたします。私が抱いたすべての疑問に対しては納得のいく回答がなされています。

署名欄：＿＿＿＿＿＿＿＿　**氏名**：＿＿＿＿＿＿＿＿

会社名：（あてはまる場合のみ）＿＿＿＿＿＿＿＿

日付：＿＿＿＿＿＿＿＿　**Turbo C シリアル番号**＿＿＿＿＿＿＿＿

ボーランドからの製品の出荷をもって、本契約の承諾に代えさせていただきます。

Turbo C ランタイム・ライブラリ・ソースコードのご注文に際しては、この裏側のページをお読みください。

Turbo C®

リファレンスガイド

バージョン 2.0

(NEC PC-9801 シリーズ)
(IBM PC, XT, AT, PS/2, および完全な互換機)

Borland International
1800 Green Hills Road
P.O. Box 660001
Scotts Valley, CA 95066-0001

編訳 (株)マイクロソフトウェア アソシエイツ 1988

目　次

はじめに

このマニュアルは，Turbo C パッケージの2冊目のマニュアル **『Turbo C リファレンスガイド』**で，Turbo C のライブラリルーチン，コモン変数，コモン型のすべての定義を含んでおり，さらにプログラム例を示して，各ルーチン，各変数，各型をどのように使えばよいかについて説明してあります。

C でのプログラミングが初めての場合は，Turbo C の1冊目のマニュアル **『Turbo C ユーザーズガイド』**をまず先に読んでください。ユーザーズガイドには，Turbo C を使用するシステムでどのようにインストールするか，また Turbo C でプログラミングを始める際に助けになる入門者向けの章が用意されています。ユーザーズガイドには，C 言語が Turbo C においてどのように実現されているかについての詳細も述べられていますし，少し高度なプログラミングテクニックについても触れられています。Turbo Pascal および Turbo Prolog をすでに使っているユーザが，Turbo C を理解しやすくするための章も設けられています。

とにかく，まずユーザーズガイドの最初の章「**はじめに**」に述べられている Turbo C 処理系に関する情報，ユーザーズガイドの内容の要約，簡単な参考文献リストを読んでください。

分冊 II：リファレンスガイド

このTurbo Cリファレンスガイドは，Cについて一通りわかっている人のために書かれており，言語および実行時の環境に関する処理系特有の詳細について述べられています。そのあと，Turbo Cで使用できる関数の定義がABC順で与えられています。このリファレンスガイドの各章についての簡単な要約を示します。

第1章「Turbo Cライブラリルーチンを使うには」では，インクルードファイル（.h）のリストと要約，Turbo Cライブラリルーチンのカテゴリ別の一覧を示し，**main**関数について説明しています。そのあと，定義済みのグローバル変数をABC順で解説していきます。

第2章「Turbo Cライブラリ」では，Turbo Cのすべてのライブラリ関数をABC順で示し，説明を加えます。各関数に対して，その形式，インクルードファイル，関連関数，機能の説明，戻り値，可搬性が示されています。なお，テキストビデオライブラリ，グラフィックスライブラリ，BIOSサポートライブラリ，日本語処理ライブラリについては，Turbo C標準ライブラリとは別に，それぞれ節をわけて解説してあります。

付録A「コンパイラエラーメッセージ」では，すべてのエラーメッセージについて説明し，考えられる原因も示しています。

付録B「言語構文の要約」では，修正BNF（バッカスナウア記法）でTurbo Cのすべての構文が記述されています。

マニュアル表記に関する規約（書体）

マニュアルの表記に際して使用される特殊なものとして次のようなものがあります。

[ ]	書式やMS-DOSコマンドラインにおける大カッコ[ ]は，システムによって異なるオプションの入力やデータを囲むものです。そのとおりにカッコをタイプしてはいけません。
〈 〉	関数の説明における不等号は，インクルードファイルを囲むのに使われます。
Boldface	Turbo Cの関数名（**printf**など）はこのようなボールド体で表記しています。
Gothic	Turbo Cの予約語（char, switch, near, cdeclなど）はゴシック体で表わします。
Italic	関数に渡す引数名（*string*, *path*など）はイタリック体で表記します。
Bold-Italic	Turbo Cで定義されているグローバル変数（***_fmode***, ***errno***など）はこの書体で表記します。
KEYCAPS	キーボード上のキー（***F10***, ***GRPH-X***, ***DEL***など）はこの書体で表記します。

謝辞

このマニュアルでは，以下に示すいくつかの製品を参照しています。

■ Turbo C，Turbo Pascal，Turbo Prolog は，Borland International 社の登録商標です。

■ PC-9801は，日本電気株式会社の商標です。

■ IBM PC, XT, および AT は，米国 International Business Machines 社の登録商標です。

■ MS-DOS, XENIX は，米国 Microsoft 社の登録商標です。

■ UNIX は，米国 American Telephone and Telegraph 社の登録商標です。

製品に関するお問い合わせ

ソフトウェアおよびマニュアルに関して，ご質問・ご意見などがありましたら，下記の**MSAカスタマーサポートセンター**まで，お電話，書面またはファクシミリにてお問い合わせください。なお，お電話での受付時間は平日の9：00am～12：00amおよび1：00pm～5：00pmとさせていただきます（土曜，日曜，祝日は休業とさせていただきますので御了承ください）。

〒107　東京都港区南青山7-8-1 小田急南青山ビル
（株）マイクロソフトウェアアソシエイツ
カスタマーサポートセンター
（TEL）03-486-1403
（FAX）03-486-8905

お問い合わせの際には，下記の事項をお伝えください。

■製品名とバージョン番号（Turbo C 2.0など）
■製品のシリアル番号（マスターディスクに記載されているもの）
■コンピュータ名とモデル番号（PC-9801VXなど）
■オペレーティングシステムとバージョン番号（MS-DOS 3.1など）

なお，書面にてお問い合わせの際には，CONFIG.SYSおよびAUTOEXEC.BATの内容，簡潔なプログラムリストを添付されるようお願い致します。

第1章

Turbo C ライブラリルーチンを使う

Turbo C には，450個以上のライブラリルーチン（関数とマクロ）が含まれています。C のプログラムの中からこれを呼び出すことができるわけで，じつにさまざまな種類の仕事を行なうことができます。低レベルから高レベルまでの入出力，文字列あるいはファイルに対する操作，メモリ割り当て，プロセス制御，データ変換，数学的計算，その他たくさんのことができるのです。

Turbo C のルーチンは，ライブラリファイル（C*x*.LIB，MATH*x*.LIB，および GRAPHICS.LIB）に含まれています。Turbo C では6種類のメモリモデルが用意されているので，タイニィを除く各メモリモデルは，それぞれ独自の（各メモリモデル用に書かれたルーチンも含めて）ライブラリファイルと数学ライブラリファイルを持っています（タイニィモデルはスモールのライブラリファイルを使用します）。

Turbo C は ANSI の C 標準化案をサポートしており，C のプログラムの中で使用される関数に対して，関数プロトタイプを宣言することができます。Turbo C の関数すべてについて，プロトタイプが1個または複数のヘッダファイルの中で宣言されています（ヘッダファイルは，.h ファイルやインクルードファイルと呼ばれるもので，INSTALL プログラムによってマスターディスクからサブディレクトリ INCLUDE にコピーされるものです）。

この章では……

このTurbo C リファレンスガイドの最初の部分では，Turbo C のライブラリルーチンとインクルードファイルを概観し，**main** 関数の働き，グローバル変数について説明します。

■どんな場合に Turbo C ランタイムライブラリのソースコードが必要になるかについて説明します。

■インクルードファイルの一覧表を示し，各々について説明を加えます。

■行なう仕事の種類によってライブラリルーチンを分類します。

■ **main** 関数に渡される引数と，その戻り値について説明します。

■ライブラリルーチンの多くで使用されるコモングローバル変数を，ABC 順で解説します。

ライブラリルーチン・リファレンス

このリファレンスガイドの第2章は，ライブラリルーチンのABC順のリファレンスであり，Turbo C ルーチンのすべてについて解説しています。

いくつかのルーチンは，似通った仕事あるいは深く関連した仕事を行なうために，“ファミリィ”としてまとめられています(たとえば，プログラムを作成，ロード，および実行する **exec**...や **spawn**...)。

そうでない関数については，必らず1つの項目をとってあります。たとえば，**free** という名前の関数について知りたければ，**free** の項を見ます。そこには以下のようなことが書かれています。

■ **free** が行なうことの要約
■ **free** を呼び出すための形式（構文）
■ **free** に対するプロトタイプを含んでいるヘッダファイルの名前
■ **free** がどのようにインプリメントされているか，および他のメモリ割り当て関数とどのように関連しているかに関する解説
■同様な関数を含んでいるたのシステムのリスト
■関連のある他の Turbo C 関数
■ものによっては，その関数をどのように使えばよいかを示すサンプルプログラム，あるいはサンプルがどこにのっているか

このリファレンスガイドの最後にある付録では，すべてのコンパイラエラーメッセージの説明，および言語構文の要約が示されています。

Turbo C ランタイムライブラリ・ソースコード

Turbo C ランタイムライブラリには，広範な領域をカバーする300以上の関数が含まれています。PC の低レベルでの制御，DOS とのインターフェース，入力/出力，プロセス管理，文字列およびメモリ操作，数学計算，ソート/検索，などです。ランタイムライブラリのソースコードが欲しくなる理由はいくつか考えられます。

■ Turbo C のある関数が，自分が書きたい関数に似てはいるけれども，まったくピッタリというわけではないことがあります。ランタイムライブラリ・ソースコードがあれば，欲しい関数を丸々書かなくても，ライブラリ関数を必要に応じて加工することができます。

■コードをデバッグしているときに，ライブラリ関数の内部についてより詳しく知りたくなることがあります。こんなときには，ランタイムライブラリ・ソースコードが大きな助けとなるでしょう。

■ C のシンボルに下線をつける慣例が気にくわず，下線がついていないバージョンのライブラリが欲しいという人がいるかもしれません。こんな場合も，ランタイムライブラリ・ソースコードがあれば，下線を取り除いたライブラリを作ることができます。

■また，プロフェッショナルが書いた緻密なライブラリソースコードから，学び取れることがたくさんあるでしょう。

こうした理由から，あるいはまた別の理由から，Turbo C ランタイムライブラリ・ソースコードが必要になるでしょう。ボーランドは，“オープンアーキテクチャ”という考えを強く持っているので，ライセンス契約によって，ユーザが Turbo C ランタイムライブラリ・ソースコードを利用できるシステムをとっています。ユーザーズガイドの最初についている注文書に必要な事項を記入して送付し，所定の方法で費用をお支払いただければ，Turbo C ランタイムライブラリ・ソースコードがお手もとに届けられます。詳しくは MSA サポートセンターまで問い合せてください。

Turbo C インクルードファイル

ヘッダファイルでは，ライブラリ関数の関数プロトタイプが宣言されています。また，ライブラリ関数が使用するデータ型やシンボリック定数の定義，Turbo C やライブラリ関数が定義しているグローバル変数も含まれています。Turbo C のヘッダファイル名とその内容は，ANSI の C 標準案に従っています。以下のリストでは，ANSI C で定義されているヘッダファイルにはアスタリスク（*）がつけられています。

alloc.h	メモリ管理関数（割り当て，解放など）を宣言しています。
assert.h *	**assert** デバッギングマクロを宣言しています。
bios.h	IBM PC の ROM ルーチンの呼び出しに使用されるさまざまな関数が宣言されています（IBM PC のみ）。
bios98.h	NEC PC-9801の ROM ルーチンの呼び出しに使用されるさまざまな関数やデータ型が宣言されています（PC-9801のみ）。
conio.h	直接コンソール I/O ルーチンを呼び出すのに使われる関数を宣言しています。
ctype.h *	文字の分類，および変換マクロ（**isalpha** や **toascii**）によって使われる情報を含んでいます。
dir.h	ディレクトリやパス名を取り扱う場合に必要となる構造体，マクロ，関数を含んでいます。
dos.h	MS-DOS や8086に特有の呼び出しに必要な定数を定義し，宣言を行なっています。
errno.h *	エラーコードに対応するニーモニック定数を定義しています。
fnctl.h	ライブラリルーチン **open** と関連して使用されるシンボリック定数を定義しています。
float.h *	浮動小数点ルーチン用のパラメータを含んでいます。
graphics.h	グラフィックス関数のプロトタイプを宣言しています。
io.h	低レベルの入力/出力ルーチンに対する構造体および宣言を含んでいます。
jctype.h	日本語処理関数が使用する文字分類テーブルなどの情報が含まれています（PC-9801のみ）。
jstring.h	日本語対応の文字列操作ルーチンのプロトタイプが宣言されています（PC-9801のみ）。
limits.h *	環境パラメータ，コンパイル時の制限値に関する情報，各種整数の値の許

される範囲などを含んでいます。

math.h ＊　数学関数のプロトタイプを宣言し，HUGE_VAL マクロを定義し，**math-err** や **_matherr** ルーチンによって使用される **exeption** 構造体を宣言しています。

mem.h　メモリ操作関数を宣言しています(その多くは，string.h の中でも定義されています)。

music.h　PC-9801のサウンドボードを制御する関数や関連するデータ型を宣言しています（PC-9801のみ)。

process.h　**spawn...** および **exec...**関数用の構造体と宣言を含んでいます。

setjmp.h ＊　**longjmp** および **setjmp** 関数によって使われる *jmp_buf* 型を定義し，**lon-gjmp** と **setjmp** のプロトタイプを宣言しています。

share.h　ファイルの共有を行なう関数で使用されるパラメータを定義しています。

signal.h ＊　**signal** と **raise** 関数が使用する定数や宣言を定義しています。

stdarg.h ＊　引数の個数が可変と宣言された関数(**vprintf**，**vscanf** など)で，引数リストを読むために使われるマクロを定義しています。

stddef.h ＊　一般的に使われるデータ型やマクロを定義しています。

stdio.h ＊　カーニハンとリッチィによって定義され，UNIX システム上で拡張された，標準入出力パッケージに必要な型とマクロを定義しています。また，標準 I/O ストリーム *stdin*，*stdout*，*stderr* を定義し，ストリームレベル I/O ルーチンを宣言しています（以降カーニハンとリッチィは K&R と略します)。

stdlib.h ＊　一般的に使用されるルーチン(変換ルーチン，サーチ/ソートルーチン，その他）を宣言しています。

string.h ＊　文字列操作およびメモリ操作ルーチンを宣言しています。

sys￥stat.h　ファイルをオープンしたり，作成したりするのに使われるシンボリック定数を定義しています。

sys￥timeb.h　**ftime** 関数と，**ftime** が返す構造体 **timeb** を宣言しています。

sys￥types.h　時刻関数とともに使用される *time_t* 型を宣言しています。

time.h ＊　時刻変換ルーチン **asctime**，**localtime**，**gmtime** で埋められる構造体，および **ctime**，**difftime**，**gmtime**，**localtime** によって使われる型を定義しています。これらのルーチンのプロトタイプも宣言されています。

values.h　重要な定数を（マシン依存性も含めて）定義しています。UNIX システム V との互換性のために提供されています。

ライブラリルーチンの分類

Turbo C ライブラリルーチンは，さまざまな種類のタスクを行ないます。この節では，ルーチンの一覧表と，そのルーチンが宣言されているインクルードファイルも合わせて示しています。

文字分類ルーチン：

ASCII 文字を，英字，コントロール文字，区切り文字，英大文字，などに分類するものです。

```
isalnum   (ctype.h)      isdigit   (ctype.h)      ispunct   (ctype.h)
isalpha   (ctype.h)      isgraph   (ctype.h)      isspace   (ctype.h)
isascii   (ctype.h)      islower   (ctype.h)      isupper   (ctype.h)
iscntrl   (ctype.h)      isprint   (ctype.h)      isxdigit  (ctype.h)
```

変換ルーチン：

文字や文字列を変換するものです。文字列から数値(浮動小数点数，整数，倍長整数)への変換や，その逆の変換，英大文字から，英小文字への変換やその逆，といったものです。

```
atof      (stdlib.h)     itoa      (stdlib.h)     tolower   (ctype.h)
atoi      (stdlib.h)     ltoa      (stdlib.h)     _tolower  (ctype.h)
atol      (stdlib.h)     strtod    (stdlib.h)     toupper   (ctype.h)
ecvt      (stdlib.h)     strtol    (stdlib.h)     _toupper  (ctype.h)
fcvt      (stdlib.h)     strtoul   (stdlib.h)     ultoa     (stdlib.h)
gcvt      (stdlib.h)     toascii   (ctype.h)
```

ディレクトリ操作ルーチン：

ディレクトリやパス名を操作するものです。

```
chdir       (dir.h)      getcurdir   (dir.h)      rmdir       (dir.h)
findfirst   (dir.h)      getcwd      (dir.h)      searchpath  (dir.h)
findnext    (dir.h)      getdisk     (dir.h)      setdisk     (dir.h)
fnmerge     (dir.h)      mkdir       (dir.h)
fnsplit     (dir.h)      mktemp      (dir.h)
```

診断ルーチン：

組み込みのエラー処理ルーチンです。

```
assert   (assert.h)      matherr   (math.h)      perror   (errno.h)
```

グラフィックスルーチン：

画面上にテキストを含むグラフィックスを作成するルーチンです。

arc	(graphics.h)	graphresult	(graphics.h)
bar	(graphics.h)	imagesize	(graphics.h)
bar3d	(graphics.h)	initgraph	(graphics.h)
circle	(graphics.h)	installuserdriver	(graphics.h)
cleardevice	(graphics.h)	installuserfont	(graphics.h)
clearviewport	(graphics.h)	line	(graphics.h)
closegraph	(graphics.h)	linerel	(graphics.h)
detectgraph	(graphics.h)	lineto	(graphics.h)
drawpoly	(graphics.h)	moverel	(graphics.h)
ellipse	(graphics.h)	moveto	(graphics.h)
fillellipse	(graphics.h)	outtext	(graphics.h)
fillpoly	(graphics.h)	outtextxy	(graphics.h)
floodfill	(graphics.h)	pieslice	(graphics.h)
getarccoords	(graphics.h)	putimage	(graphics.h)
getaspectratio	(graphics.h)	putpixel	(graphics.h)
getbkcolor	(graphics.h)	rectangle	(graphics.h)
getcolor	(graphics.h)	registerbgidriver	(graphics.h)
getdefaultpalette	(graphics.h)	registerbgifont	(graphics.h)
getdrivername	(graphics.h)	restorecrtmode	(graphics.h)
getfillpattern	(graphics.h)	sector	(graphics.h)
getfillsettings	(graphics.h)	setactivepage	(graphics.h)
getgraphmode	(graphics.h)	setallpalette	(graphics.h)
getimage	(graphics.h)	setaspectratio	(graphics.h)
getlinesettings	(graphics.h)	setbkcolor	(graphics.h)
getmaxcolor	(graphics.h)	setcolor	(graphics.h)
getmaxmode	(graphics.h)	setfillpattern	(graphics.h)
getmaxx	(graphics.h)	setfillstyle	(graphics.h)
getmaxy	(graphics.h)	setgraphbufsize	(graphics.h)
getmodename	(graphics.h)	setgraphmode	(graphics.h)
getmoderange	(graphics.h)	setlinestyle	(graphics.h)
getpalette	(graphics.h)	setnewdriver	(graphics.h)
getpalettesize	(graphics.h)	setpalette	(graphics.h)
getpixel	(graphics.h)	setrgbpalette	(graphics.h)
gettextsettings	(graphics.h)	settextjustfy	(graphics.h)
getviewsettings	(graphics.h)	settextstyle	(graphics.h)
getx	(graphics.h)	setusercharsize	(graphics.h)
gety	(graphics.h)	setviewport	(graphics.h)
graphdefaults	(graphics.h)	setvisualpage	(graphics.h)
grapherrormsg	(graphics.h)	setwritemode	(graphics.h)
_graphfreemem	(graphics.h)	textheight	(graphics.h)
_graphgetmem	(graphics.h)	textwidth	(graphics.h)

入力/出力ルーチン：

ストリームレベル，およびDOSレベルのI/O機能を提供するルーチンです。

access	(io.h)	fputc	(stdio.h)	putw	(stdio.h)
cgets	(conio.h)	fputchar	(stdio.h)	_read	(io.h)
_chmod	(io.h)	fputs	(stdio.h)	read	(io.h)
chmod	(io.h)	fread	(stdio.h)	remove	(stdio.h)
chsize	(io.h)	freopen	(stdio.h)	rename	(stdio.h)
clearerror	(stdio.h)	fscanf	(stdio.h)	rewind	(stdio.h)
close	(io.h)	fseek	(stdio.h)	scanf	(stdio.h)
_close	(io.h)	fsetpos	(stdio.h)	setbuf	(stdio.h)
cprintf	(conio.h)	fstat	(sys¥stat.h)	setftime	(io.h)
cputs	(conio.h)	ftell	(stdio.h)	setmode	(io.h)
creat	(io.h)	ffwrite	(stdio.h)	setvbuf	(stdio.h)
_creat	(io.h)	getc	(stdio.h)	sopen	(io.h)
creatnew	(io.h)	getch	(conio.h)	sprintf	(stdio.h)
creattemp	(io.h)	getchar	(stdio.h)	sscanf	(stdio.h)
cscanf	(conio.h)	getche	(conio.h)	stat	(sys¥stat.h)
dup	(io.h)	getftime	(io.h)	_strerror	(string.h, stdio.h)
dup2	(io.h)	getpass	(conio.h)		
eof	(io.h)	gets	(stdio.h)	strerror	(string.h)
fclose	(stdio.h)	getw	(stdio.h)	tell	(io.h)
fcloseall	(stdio.h)	ioctl	(io.h)	tmpfile	(stdio.h)
fdopen	(stdio.h)	isatty	(io.h)	tmpnam	(stdio.h)
feof	(stdio.h)	kbhit	(conio.h)	ungetc	(stdio.h)
ferror	(stdio.h)	lock	(io.h)	ungetch	(conio.h)
fflush	(stdio.h)	lseek	(io.h)	unlock	(io.h)
fgetc	(stdio.h)	_open	(io.h)	vfprintf	(stdio.h)
fgetchar	(stdio.h)	open	(io.h)	vfscanf	(stdio.h)
fgetpos	(stdio.h)	perror	(stdio.h)	vprintf	(stdio.h)
fgets	(stdio.h)	printf	(stdio.h)	vscanf	(stdio.h)
filelength	(io.h)	putc	(stdio.h)	vsprintf	(stdio.h)
fileno	(stdio.h)	putch	(stdio.h)	vsscanf	(stdio.h)
flushall	(stdio.h)	putchar	(stdio.h)	_write	(stdio.h)
fopen	(stdio.h)	puts	(stdio.h)	write	(stdio.h)
fprintf	(stdio.h)				

インターフェースルーチン（MS-DOS，8086）：

MS-DOS，8086など，マシン特有の機能を与えます。

absread	(dos.h)	getfatd	(dos.h)	outport	(dos.h)
abswrite	(dos.h)	getpsp	(dos.h)	outportp	(dos.h)
bdos	(dos.h)	getvect	(dos.h)	parsfnm	(dos.h)
bdosptr	(dos.h)	getverify	(dos.h)	peek	(dos.h)
country	(dos.h)	harderr	(dos.h)	peekb	(dos.h)
ctrlbrk	(dos.h)	hardresume	(dos.h)	poke	(dos.h)
disable	(dos.h)	hardretn	(dos.h)	pokeb	(dos.h)
dosexterr	(dos.h)	inport	(dos.h)	randbrd	(dos.h)
enable	(dos.h)	inportb	(dos.h)	randbwr	(dos.h)
FP_OFF	(dos.h)	int86	(dos.h)	segread	(dos.h)
FP_SEG	(dos.h)	int86x	(dos.h)	setcbrk	(dos.h)
freemem	(dos.h)	intdos	(dos.h)	setdta	(dos.h)
geninterrupt	(dos.h)	intdosx	(dos.h)	setvect	(dos.h)
getcbrk	(dos.h)	intr	(dos.h)	setverify	(dos.h)
getdfree	(dos.h)	keep	(dos.h)	sleep	(dos.h)
getdta	(dos.h)	MK_FP	(dos.h)	unlink	(dos.h)
getfat	(dos.h)				

文字列・メモリ操作ルーチン：

文字列あるいはメモリブロックを操作するものです。コピー，比較，変換，探索などを行ないます。

memccpy	(mem.h, string.h)	strchr	(string.h)	strncmpi	(string.h)
memchr	(mem.h, string.h)	strcmp	(string.h)	strncpy	(string.h)
memcmp	(mem.h, string.h)	strcmpi	(string.h)	strnicmp	(string.h)
memcpy	(mem.h, string.h)	strcpy	(string.h)	strnset	(string.h)
memicpy	(mem.h, string.h)	strcspn	(string.h)	strpbrk	(string.h)
memmove	(mem.h, string.h)	strdup	(string.h)	strrchr	(string.h)
memset	(mem.h, string.h)	strerror	(string.h)	strrev	(string.h)
movedata	(mem.h, string.h)	stricmp	(string.h)	strset	(string.h)
movmem	(mem.h, string.h)	strlen	(string.h)	strspn	(string.h)
setmem	(mem.h)	strlwr	(string.h)	strstr	(string.h)
stpcpy	(string.h)	strncat	(string.h)	strtok	(string.h)
strcat	(string.h)	strncmp	(string.h)	strupr	(string.h)

数学ルーチン：

数学的計算や変換を行なうものです。

```
abs          (stdlib.h)
acos         (math.h)
asin         (math.h)
atan         (math.h)
atan2        (math.h)
atof         (math.h, stdlib.h)
atoi         (stdlib.h)
atol         (stdlib.h)
cabs         (math.h)
ceil         (math.h)
_clear87     (float.h)
_control87   (float.h)
cos          (math.h)
cosh         (math.h)
div          (stdlib.h)
ecvt         (stdlib.h)
exp          (math.h)
fabs         (math.h)
fcvt         (stdlib.h)
floor        (math.h)
fmod         (math.h)
_fpreset     (float.h)
frexp        (math.h)
gcvt         (stdlib)
hypot        (math.h)
itoa         (stdlib.h)
labs         (stdlib.h)
ldexp        (math.h)
ldiv         (stdlib.h)
log          (math.h)
log10        (math.h)
_lrotl       (stdlib.h)
_lrotr       (stdlib.h)
ltoa         (stdlib.h)
_matherr     (math.h)
matherr      (math.h)
modf         (math.h)
poly         (math.h)
pow          (math.h)
pow10        (math.h)
rand         (stdlib.h)
random       (stdlib.h)
randomize    (stdlib.h)
_rotl        (stdlib.h)
_rotr        (stdlib.h)
sin          (math.h)
sinh         (math.h)
sqrt         (math.h)
srand        (stdlib.h)
_status87    (float.h)
strtod       (stdlib.h)
strtol       (stdlib.h)
strtoul      (stdlib.h)
tan          (math.h)
tanh         (math.h)
ultoa        (stdlib.h)
```

メモリ割り当てルーチン：

スモールデータモデルおよびラージデータモデルで，動的メモリ割り当てを行なうルーチンです。

```
allocmem     (dos.h)
brk          (alloc.h)
calloc       (alloc.h)
coreleft     (alloc.h, stdlib.h)
farcalloc    (alloc.h)
farcoreleft  (alloc.h)
farfree      (alloc.h)
farmalloc    (alloc.h)
farrealloc   (alloc.h)
free         (alloc.h, stdlib.h)
malloc       (alloc.h, stdlib.h)
realloc      (alloc.h, stdlib.h)
sbrk         (alloc.h)
setblock     (dos.h)
```

その他のルーチン：

ローカルでない goto 機能や，ビープ音の制御などを与えます（*は PC-9801のみの関数です）。

```
beep *         (dos.h)
delay          (dos.h)
getfont *      (dos.h)
longjmp        (setjmp.h)
nosound        (dos.h)
putuserfont *  (dos.h)
setjmp         (setjmp.h)
sound          (dos.h)
```

プロセス制御ルーチン：

あるプロセスの中から，新しいプロセスを呼び出したり，終了させたりするものです。

abort	(process.h)	raise	(signal.h)
execl	(process.h)	signal	(signal.h)
execle	(process.h)	spawnl	(process.h)
execlp	(process.h)	spawnle	(process.h)
execlpe	(process.h)	spawnlp	(process.h)
execv	(process.h)	spawnlpe	(process.h)
execve	(process.h)	spawnv	(process.h)
execvp	(process.h)	spawnve	(process.h)
execvpe	(process.h)	spawnvp	(process.h)
_exit	(process.h)	spawnvpe	(process.h)
exit	(process.h)	system	(process.h)

標準ルーチン：

これらは標準ルーチンです。

abort	(stdlib.h)	gcvt	(stdlib.h)	random	(stdlib.h)
abs	(stdlib.h)	getenv	(stdlib.h)	randomize	(stdlib.h)
atexit	(stdlib.h)	itoa	(stdlib.h)	realloc	(stdlib.h)
atof	(stdlib.h)	labs	(stdlib.h)	_rotl	(stdlib.h)
atoi	(stdlib.h)	lfind	(stdlib.h)	_rotr	(stdlib.h)
atol	(stdlib.h)	_lrotl	(stdlib.h)	srand	(stdlib.h)
bsearch	(stdlib.h)	_lrotr	(stdlib.h)	strtod	(stdlib.h)
calloc	(stdlib.h)	lsearch	(stdlib.h)	strtol	(stdlib.h)
ecvt	(stdlib.h)	ltoa	(stdlib.h)	strtoul	(stdlib.h)
_exit	(stdlib.h)	malloc	(stdlib.h)	swab	(stdlib.h)
exit	(stdlib.h)	putenv	(stdlib.h)	system	(stdlib.h)
fcvt	(stdlib.h)	qsort	(stdlib.h)	ultoa	(stdlib.h)
free	(stdlib.h)	rand	(stdlib.h)		

テキストウィンドウ表示ルーチン：

テキストウィンドウの制御と出力を行なうルーチンです（＊は PC-9801のみの関数です）。

clreol	(conio.h)	movetext	(conio.h)	textcursor *	(conio.h)
clrscr	(conio.h)	normvideo	(conio.h)	textmode	(conio.h)
delline	(conio.h)	puttext	(conio.h)	textreverse *	(conio.h)
gettext	(conio.h)	textattr	(conio.h)	textunder *	(conio.h)
gettextinfo	(conio.h)	textbackground	(conio.h)	textvertical *	(conio.h)
gotoxy	(conio.h)	textbank *	(conio.h)	wherex	(conio.h)
highvideo	(conio.h)	textblink *	(conio.h)	wherey	(conio.h)
insline	(conio.h)	textcolor	(conio.h)	window	(conio.h)
lowvideo	(conio.h)				

時刻および日付ルーチン：

時刻変換，時刻操作のルーチンです。

asctime	(time.h)	getdate	(dos.h)	settime	(dos.h)
ctime	(time.h)	gettime	(dos.h)	stime	(time.h)
difftime	(time.h)	gmtime	(time.h)	time	(time.h)
dostounix	(dos.h)	localtime	(time.h)	tzset	(time.h)
ftime	(sys¥timeb.h)	setdate	(dos.h)	unixtodos	(dos.h)

変数引数リストルーチン：

vprintf などの中で，変数引数のリストにアクセスする場合に使います。

va_arg	(stdarg.h)	va_end	(stdarg.h)	va_start	(stdarg.h)

BIOS インターフェース（PC-9801）：

PC-9801の ROM-BIOS を呼び出すルーチンです。

bios98com	(bios98.h)	bios98key	(bios98.h)
bios98com_ch2	(bios98.h)	bios98memory	(bios98.h)
bios98com_ch3	(bios98.h)	bios98mouse	(bios98.h)
bios98com_init	(bios98.h)	bios98mouse_init	(bios98.h)
bios98com_init_ch2	(bios98.h)	bios98msw	(bios98.h)
bios98com_init_ch3	(bios98.h)	bios98print	(bios98.h)
bios98disk	(bios98.h)	bios98stoptimer	(bios98.h)
bios98equip	(bios98.h)	bios98time	(bios98.h)
bios98harddisk	(bios98.h)	bios98timer	(bios98.h)

BIOS インターフェース（IBM PC）：

IBM PC の ROM-BIOS を呼び出すルーチンです。

bioscom	(bios.h)	bioskey	(bios.h)	biosptint	(bios.h)
biosdisk	(bios.h)	biosmemory	(bios.h)	biostime	(bios.h)
biosequip	(bios.h)				

日本語処理ルーチン（PC-9801のみ）：

日本語対応の文字列操作および文字分類ルーチンです。

btom	(jstring.h)	jishira	(jtype.h)	jstrmatch	(jstring.h)
chkctype	(jctype.h)	jiskana	(jtype.h)	jstrncat	(jstring.h)
hantozen	(jctype.h)	jiskata	(jtype.h)	jstrncmp	(jstring.h)
isalkana	(jctype.h)	jiskigou	(jtype.h)	jstrncpy	(jstring.h)
isalnmkana	(jctype.h)	jisl0	(jtype.h)	jstrrchr	(jstring.h)
isgrkana	(jctype.h)	jisl1	(jtype.h)	jstrrev	(jstring.h)
iskana	(jctype.h)	jisl2	(jtype.h)	jstrskip	(jstring.h)
iskanji	(jctype.h)	jislower	(jtype.h)	jstrstr	(jstring.h)
iskanji2	(jctype.h)	jisspace	(jtype.h)	jstrtok	(jstring.h)
iskmoji	(jctype.h)	jistojms	(jtype.h)	jtohira	(jtype.h)
iskpun	(jctype.h)	jisupper	(jtype.h)	jtokana	(jtype.h)
ispnkana	(jctype.h)	jiszen	(jtype.h)	jtokata	(jtype.h)
isprkana	(jctype.h)	jmstojis	(jtype.h)	jtolower	(jtype.h)
jasctime	(time.h)	jstradv	(jstring.h)	jtoupper	(jtype.h)
jctime	(time.h)	jstrchr	(jstring.h)	mtob	(jstring.h)
jisalpha	(jtype.h)	jstrcmp	(jstring.h)	nthctype	(jtype.h)
jisdigit	(jtype.h)	jstrlen	(jstring.h)	zentohan	(jtype.h)

サウンドライブラリ（PC-9801のみ）：

PC-9801のサウンドボードを操作するルーチンです。

mc_block	(music.h)	mc_inquire	(music.h)	mc_rom	(music.h)
mc_continue	(music.h)	mc_mode	(music.h)	mc_scalar	(music.h)
mc_ground	(music.h)	mc_play	(music.h)	mc_stop	(music.h)
mc_initialize	(music.h)	mc_register	(music.h)		

main 関数

Cのプログラムには，すべて **main** 関数がなければなりません（それをどこに置くかは，好みの問題ですが)。**main** をファイルの先頭に置くプログラマもあり，一番最後に置く人もいます。しかし，置かれる位置とは関係なく，**main** 関数に対しては，以下に示すようなポイントが必ず適用されます。

main への引数

ここに示すのは，Turbo C スタートアップルーチンによって **main** に渡されるパラメータ（引数)，***argc***，***argv***，および ***env*** です。

■整数 ***argc*** は，**main** に渡されるコマンドライン引数の個数です。

■ ***argv*** は文字列へのポインタの配列です。

・バージョン3.*x* の MS-DOS では，***argv***[0]は実行されるプログラムのフルパスとして定義されます。

・バージョン3.0未満の MS-DOS の下では，***argv***[0]はヌル文字列（""）を指します。

・ ***argv***[1]は，MS-DOS のコマンドラインでプログラム名の後に最初にタイプされた文字列を指しています。

・ ***argv***[2]は，プログラム名の2つ後にタイプされた文字列を指しています。

・ ***argv***[***argc***-1]は，**main** に渡された最後の引数を指しています。

・ ***argv***[***argc***]は NULL になります。

■ ***env*** も文字列へのポインタの配列です。***env***[]の各要素は，*ENVVAR*＝*value* の形式の文字列を指しています。

・*ENVVAR* は環境変数の名前で，PATH，87などがあります。

・*value* は *ENVVAR* にセットされる値で，たとえばPATH には A：¥DOS;A：¥TURBOC などが，87には YES などが与えられます。

Turbo C スタートアップルーチンは，必らずこの三つの引数を **main** に渡しますが，プログラムの中でこれらを定義するかどうかはプログラマが決めることができます。これらの引数の内のいくつか（あるいはすべて）を宣言すれば，**main** ルーチン内でローカル変数として使えるようになります。

ただし，これらのパラメータを宣言する場合は，必らず，***argc***，***argv***，***env***，の順で宣言しなければならないことに注意してください。

以下に示す例は，**main** の引数の宣言として，いずれも有効です。

```
main()
main(int argc)                          /* 有効だが一般的ではない */
main(int argc, char * argv[])
main(int argc, char * argv[], char * env[])
```

注意 1：二番目の宣言 **main**(int *argc*) は誤りではありませんが，プログラム内で *argv* の要素は使わずに *argc* を使うということで，あまり一般的ではありません。

注意 2：引数 *env* は，グローバル変数 ***environ*** を通してでも使用することができます。この章の ***environ*** の項，および第2章の **putenv** と **getenv** の項を見れば，より詳しい情報が得られます。

argc, argv, env を使用するサンプルプログラム

次に示す例は，**main** に渡されるこれらの引数を利用する方法を示すプログラム（ARGS.EXE）です。

```
プログラム ARGS.EXE */

include <stdio.h>
include <stdlib.h>

main(int argc, char *argv[], char *env[])
{
    int i;

    printf("The value of argc is %d ¥n¥n",argc);
    printf("These are the %d command-line arguments passed to main: ¥n¥n",
            argc);

    for (i = 0; i <= argc; i++)
        printf("   argv[%d]: %s¥n", i, argv[i]);

    printf("¥nThe environment string(s) on this system are:¥n¥n");

    for (i = 0; env[i] != NULL; i++)
        printf("   env[%d]: %s¥n", i, env[i]);
}
```

このプログラム ARGS.EXE を，MS-DOS のプロンプトから次のようなコマンドラインで実行したとします。

```
A:¥> args first_argument "argument with blanks" 3 4 "last but one" stop!
```

この例の"argument with blanks"や"last but one"のように，二重引用符で囲めば，空白を含む引数も渡せることに注意してください。

ARGS.EXE の出力は，（環境変数がここに示すように設定されているとすれば）次のようになります。

```
The value of argc is 7

These are the 7 command-line arguments passed to main:

    argv[0]: A:¥TURBOC¥ARGS.EXE
    argv[1]: first_argument
    argv[2]: argument with blanks
    argv[3]: 3
    argv[4]: 4
    argv[5]: last but one
    argv[6]: stop!
    argv[7]: (null)

The environment string(s) on this system are:

    env[0]: COMSPEC=A:¥COMMAND.COM
    env[1]: PROMPT=$p$g
    env[2]: PATH=A:¥SPRINT;A:¥DOS;A:¥TURBOC
```

注意：**main** に渡されるコマンドライン全体の最大の長さは（引数の区切りの空白も含めて）128文字になります。これは MS-DOS の制限です。

main へのワイルドカード引数

ワールドカード文字を含むコマンドライン引数は，DOS が COPY コマンドに渡されたワイルドカードを展開するのとまったく同じ方法で，一致するすべてのファイル名に展開されるようにすることができます。ワイルドカードの展開に必要なことは，Turbo C に含まれているオブジェクトファイル WILDARGS.OBJ をプログラムにリンクするだけです。

WILDARGS.OBJ がプログラムコードにリンクされていれば，＊.＊などの形式のワイルドカード引数を **main** 関数に送ることができます。ワイルドカード引数は，（*argv* 配列の中に）ワイルカードマスクに一致するすべてのファイル名として展開されます。*argv* 配列の最大サイズは，ヒープ中で使用可能なメモリの量によって変化します。

一致するファイルが見つからない場合は，引数はそのまま渡されます（つまり，ワイルドカードを含む文字列が **main** に渡されます）。

例：次の2つのコマンドは，ファイル ARGS.C をコンパイルしてワイルドカード展開モジュール WILDARGS.OBJ をリンクし，作成された実行可能ファイル ARGS.EXE を実行したものです。

```
tcc  args  wildargs.obj

args  A:¥TURBOC¥INCLUDE¥*.H  "*.C"
```

ARGS.EXE が実行されると，最初の引数は，A：¥TURBOC¥INCLUDE ディレクトリにあるすべての.H ファイル名に展開されます。展開後の引数文字列には，完全なパスが含まれることに注意してください（たとえば A：¥TURBOC¥INCLUDE¥ALLOC.H)。2番目の引数＊.C は，引用符で囲んであるので展開されません。

統合環境(TC.EXE)の場合には，次のような内容のプロジェクトファイルを作って，そのファイル名を **P**roject/**P**roject name で指定してください。

```
ARGS
WILDARGS.OBJ
```

そして，**O**ptions/**A**rguments オプションを使って，コマンドライン引数をセットします。

注意：いちいち WILDARGS.OBJ をリンクしなくても，ワイルドカードの展開がデフォルトで可能となるようにしたい場合は，標準のライブラリファイル C*x*.LIB に WILDARGS.OBJ を組み入れてしまうことができます。これを行なうには，標準ライブラリから SETARGV モジュールを削除して，WILDARGS を追加します。次のコマンドは，Turbo ライブラリアン TLIB を呼び出して，すべての標準ライブラリを変更しています（これは，標準 C ライブラリと WILDARGS.OBJ がカレントディレクトリにある場合です)。

```
tlib  cs  -setargv  +wildargs
tlib  cc  -setargv  +wildargs
tlib  cm  -setargv  +wildargs
tlib  cl  -setargv  +wildargs
tlib  ch  -setargv  +wildargs
```

-p オプションを使ったコンパイル (Pascal の呼び出し慣例)

Pascal の呼び出し慣例（ユーザーズガイドの第12章に詳細があります）を用いてプログラムをコンパイルする場合は，**main** が C のタイプであることを明示して宣言しなければなりません。

それには，次のように予約語 **cdecl** を使います。

```
cdecl  main(int argc, char *argv[], char *envp[])
```

main が返す値

main が返す値は，そのプログラムのステータスコード(int)になります。ただし，プログラムの終了に **exit** ルーチン（あるいは_**exit**）を使うと，**main** が返す値は **exit**（あるいは_**exit**）を呼び出すときに渡された引数の値になります。

たとえば，プログラムが次のような呼び出しをしていると，ステータスは1になります。

```
exit(1)
```

Turbo C の統合環境版（TC.EXE）を使っている場合には，プログラムの実行が終了してから，Compile メニューの Get info を選べば（***GRPH-C***, ***G***），**main** からの戻り値を見ることができます。

"Function should return a value"（関数は値を返さなければならない）の警告をオンにしている場合（-wrvl または **O/C/Errors/Common errors/A**：**...** On)，そのチェックは **main** 関数にも適用されます。**main** に対してこの警告を出したくなければ，return 文で値を返すようにするか，**main** を次のように void 型と宣言するようにしてください。

```
void main()
{
    ...
```

グローバル変数

_8087

機能　　コプロセッサチップフラグです。

形式　　extern int _8087；

宣言ファイル　dos.h

解説　　変数 ***_8087*** は，スタートアップコードの自動検出ルーチンが浮動小数点コプロセッサ（8087，80287，あるいは80387）が装着されていることを検出した場合には，0でない値(チップの種類によってそれぞれ1，2，あるいは3）にセットされます。そうでない場合には，***_8087*** は0にセットされます。

自動検出機能は，環境変数87を YES または NO にセットすることによって無効にすることができます(DOS のプロンプトから SET 87=YES あるいは SET 87=NO というコマンドを与えます。等号の前後にスペースを入れてはいけません)。この場合には ***_8087*** の値は，環境変数の値にしたがって1(YES)または0(NO)にセットされます。

環境変数87については，ユーザーズガイドの第12章を参照してください。

プログラム内に浮動小数点コードが含まれている場合，***_8087*** は適切な値（1～3）にセットされていなければなりません。

_*argc*

機能　　コマンドライン引数の個数を保持しています。

形式　　extern int _*argc*；

宣言ファイル　dos.h

解説　　_***argc*** は，プログラムの実行開始時に **main** に渡された ***argc*** の値を持っています。

_*argv*

機能　　コマンドライン引数へのポインタの配列です。

形式　　extern char ＊ _*argv*[]；

宣言ファイル　dos.h

解説　　_***argv*** は，プログラムの実行開始時に **main** に渡されたもともとのコマンドライン引数（***argv***[]の要素）を持つ配列を指しています。

daylight

機能　夏時間が有効かどうかを示しています。

形式　extern int *daylight*;

宣言ファイル　time.h

解説　***daylight*** は，時刻日付関数によって使用されます。
daylight は，**tzset**, **ftime** および **localtime** 関数によって，夏時間の実施期間中であれば1，標準時間中であれば0にセットされます。

directvideo

機能　ビデオ出力を制御するフラグです。

形式　extern int *directvideo;*

宣言ファイル　time.h

解説　***directvideo*** は，プログラムのコンソール出力（**cputs** などによるもの）が，直接ビデオ RAM に書き込まれる（***directvideo*** = 1）か，ROM BIOS を通じて送られる（***directvideo*** = 0）かを制御しています。
デフォルト値は ***directvideo*** = 1で，コンソール出力はビデオ RAM に直接送られます。***disrectvideo*** = 1として使用する場合には，システムのビデオハードウェアは，IBM のディスプレイアダプタと完全な互換性がなければなりません。***directvideo*** = 0にセットしている場合には，コンソール出力は IBM BIOS とコンパチブルなすべてのシステムで動作します。

注意：PC−9801用では，***directvideo*** は宣言はされていますが，まったく使用されていません。

environ

機能　DOS の環境変数へのアクセスです。

形式　extern char ＊ *environ*[];

宣言ファイル　dos.h

解説　***environ*** は文字列へのポインタの配列で，プロセスの環境にアクセスしたり，環境を変更したりする場合に使用します。各文字列は次のような形式になっています。

```
varname = varvalue
```

varname は環境変数の名前(PATH など)，*varvalue* は環境変数(*varname*)にセットされている値(A：￥BIN;A：￥DOS など)です。文字列 *varvalue* は空の場合もあります。

プログラムの実行開始時に，MS-DOS の環境設定は，直接そのプログラムに渡されます。**main** への3番目の引数 *envp* は，***environ*** の初期設定と同じであることに注意してください。

environ 配列は，**getenv** 関数によってアクセスすることができますが，配列 ***environ*** の要素の，追加，変更，削除が行なえるのは **putenv** 関数だけです。これは，修正によってプロセス環境変数の大きさが変更されたり，再配置されたりしても，***environ*** は自動的に調整されてその配列を指しているようにするためです。

errno, *_doserrno*, *sys_errlist*, *sys_nerr*

機能　　**perror** がエラーメッセージをプリントすることを可能にします。

形式　　extern int *errno*;
extern int *_doserrno*;
extern char * *sys_errlist*;
extern int *sys_nerr*;

宣言ファイル　　errno.h, stdlib.h（***errno***, ***_doserrno***, ***sys_errlist***, ***sys_nerr***）
dos.h（***_doserrno***）

解説　　***errno***, ***sys_errlist***, ***sys_nerr*** は，ライブラリルーチンが指定されたタスクが実行できなかったときに，**perror** 関数がエラーメッセージを出力するために使用します。***_doserrno*** は，多くの DOS エラーコードを ***errno*** に対応させる変数です。ただし，**perror** が ***_doserrno*** を直接使用することはありません。

_doserrno：DOS システムコールがエラーとなった場合，***_doserrno*** には，DOS の実際のエラーコードがセットされます。***errno*** は，ほぼ同様な変数ですが，UNIX システムから引き継いでいるものです。

errno：システムコールでエラーが起こった場合，***errno*** はエラーのタイプを示すようにセットされます。***errno*** が ***_doserrno*** と同じこともありますが，***errno*** には MS-DOS のエラーコードが含まれない（***_doserrno*** には含まれている）こともあります。また逆に，***errno*** だけがセットされて，***_doserrno*** はセットされない場合もあります。

sys_errlist：メッセージの書式を制御するために，メッセージ文字列の配列が，***sys_errlist*** として与えられています。***errno*** はこの配列の添字として使われ，エラー番号に対応する文字列を見い出すことができます。文字列には改行文字は含まれていません。

sys_nerr：この変数は，***sys_errlist*** 内に含まれているエラーメッセージの個数として定義されています。

以下に示すのは，***sys_errlist*** 内に格納されている値に対するニーモニックとその意味です。

ニーモニック	意味
E2BIG	引数リストが長すぎる
EACCES	アクセス権が無い
EBADF	ファイル番号が正しくない
ECONTR	メモリブロックが壊れている
ECURDIR	カレントディレクトリを削除しようとした
EDOM	定義域エラー
EEXIST	ファイルはすでに存在している
EINVACC	アクセスコードが正しくない
EINVAL	引数が正しくない
EINVDAT	データが正しくない
EINVDRV	指定されたドライブが正しくない
EINVENV	環境が正しくない
EINVFMT	書式が正しくない
EINVFNC	ファンクション番号が正しくない
EINVMEM	メモリブロックアドレスが正しくない
EMFILE	オープンファイルが多すぎる
ENMFILE	これ以上のファイルはない
ENODEV	そのデバイスは存在しない
ENOENT	そのファイルあるいはディレクトリは存在しない
ENOEXEC	exec 書式エラー
ENOFILE	そのファイルあるいはディレクトリは存在しない
ENOMEM	メモリが足りない
ENOPATH	パスが見つからない
ENOSPC	UNIX である － DOS ではない
ENOTSAM	同一のデバイスではない
ERANGE	値域エラー(結果が指定の範囲を越えた)

EXDEV	クロスデバイスリンク
EZERO	エラー0

次に示すのは，_**doserrno** にセットされる，実際の MS-DOS のエラーコードに対するニーモニックです。_**doserrno** は，**errno** を通じて，**sys_errlist** 内のエラーメッセージ文字列に対応づけされる場合と，されない場合があります。

ニーモニック	**意味**
EINVAL	無効なファンクション
E2BIG	不正な環境
EACCES	アクセスは否定されている
EACCES	無効なアクセス
EACCES	カレントディレクトリである
EBADF	無効なハンドル
EFAULT	（予約されています）
EINVAL	無効なデータ
EMFILE	オープンされているファイルが多すぎる
ENOENT	そのようなファイルまたはディレクトリはない
ENOEXEC	不正なフォーマット
ENOMEM	メモリコントロールブロックが破壊されている
ENOMEM	メモリが不足した
ENOMEM	無効なメモリブロック
EXDEV	無効なドライブ
EXDEV	同じデバイスではない

DOS のエラーリターンコードについては，MS-DOS プログラマーズリファレンスマニュアルにより詳しい情報があります。

_fmode

機能　デフォルトのファイル変換モードを決定します。

形式　extern unsiged *_fmode*;

宣言ファイル　fcntl.h

解説　***_fmode*** は，ファイルがどのモード(テキストまたはバイナリ)でオープンされ，変換が行なわれるかを決めるものです。***_fmode*** の初期値はO TEXT であり，この場合ファイルはテキストモードで読まれます。***_fmode*** がO_BINARY にセットされると，ファイルはバイナリモードでオープンされ，読まれます(O_TEXT とO_BINARY は，fcntl.h の中で定義されています)。
テキストモードでは，入力時に，CR/LF のペアはLF1文字に変換されます。出力時にはその逆の変換が行なわれ，LF はCR/LF に変換されます。バイナリモードではこのような変換は行なわれません。

ライブラリルーチ **fopen**，**fdopen**，および **freopen** の引数 *type* に，"t"(テキストモード）または"b"（バイナリモード）を指定すれば，***_fmode*** によってセットされているデフォルトのモードは無効にすることができます。また，ライブラリルーチン **open** でも，引数 *access* にO_BINARY またはO_TEXT を指定すれば，**open** の引数 *pathname* で示されたファイルを，バイナリモード，テキストモードどちらでオープンするかを決めることができます。

_heaplen

機能　　ヒープサイズを保持しています。

形式　　extern unsigned _*heaplen*；

解説

_heaplenは，スモールデータモデル（タイニィ，スモール，ミディアム）では，near ヒープのサイズを示しています。ラージデータモデル（コンパクト，ラージ，ヒュージ）では near ヒープがないため，***_heaplen***も存在しません。

スモールおよびミディアムモデルでは，データセグメントのサイズは次のようにして計算されます。

データセグメント［スモール，ミディアム］
＝ グローバルデータ ＋ ヒープ ＋ スタック

ここで，スタックサイズは***_stklen***によって調整することができます。

_heaplenが0にセットされている場合は，プログラムはデータセグメントに64K バイトを割り当て，ヒープの実効サイズは次のようになります。

64K -（グローバルデータ ＋ スタック）バイト

デフォルトでは***_heaplen***＝0なので，***_heaplen***になんらかの値を指定しなければ，64K のデータセグメントが得られることになります。

タイニィモデルでは，コードも含めたすべてが同一のセグメントに置かれるので，データセグメントは，コードと，プログラムセグメントプレフィクス（PSP）の256バイトを含む長さに調整されます。

データセグメント ＝ 256 ＋ コード ＋ グローバルデータ
＋ ヒープ ＋ スタック

タイニィモデルで***_heaplen***＝0の場合，ヒープの実効サイズは，64Kから，PSP，コード，グローバルデータ，およびスタックを引いたものになります。

コンパクトおよびラージモデルでは，nearヒープは存在しないので，データセグメントは次のように単純に求められます。

データセグメント ＝ グローバルデータ ＋ スタック

ヒュージモデルでは，スタックは別個のセグメントにとられ，各モジュールはモジュールごとにデータセグメントを持ちます。

関連項目　***_stklen***

_osmajor, _osminor

機能　DOSのバージョン番号のメジャー部およびマイナー部を保持しています。

形式　extern unsigned char _*osmajor*;
extern unsigned char _*osminor*;

宣言ファイル　dos.h

解説　DOSバージョンのメジャー部とマイナー部は，**_osmajor**と**_osminor**によって別々に得ることができます。メジャーバージョン番号が**_osmajor**に，マイナーバージョン番号が**_osminor**に入っています。
これらの変数は，MS-DOSのバージョン2.*x*と3.*x*の両方で走るモジュールを必要とする場合に便利です。ライブラリルーチンの中には，MS-DOSのバージョンによって動作が異なるものがいくつかあり，バージョン3.xの下でしか動作しないものもあります(第2章の，**_open**，**creatnew**，**ioctl**の項を参照してみてください)。

_psp

機能　PSPのセグメントアドレスを保持しています。

形式　extern unsigned int _*psp*;

宣言ファイル　dos.h

解説　**_psp**は，現在のプログラムに対するプログラムセグメントプレフィクス（PSP)のセグメントアドレスを保持しています。PSPはDOSのプロセスディスクリプタであり，プログラムについてのDOSの初期情報を含んでいます。
PSPに関する詳細な情報については，MS-DOSプログラマーズリファレンスマニュアルを参照してください。

_stklen

名前　スタックサイズを保持しています。

形式　extern unsigned _*stklen*；

宣言ファイル　dos.h

解説　_**stklen** は，6つのメモリモデルのすべてにおいて，スタックサイズを示しています。スタックの最小サイズは128ワードで，これより小さい値を与えると，_**stklen** は自動的に規定の最小値に調整されます。デフォルトのスタックサイズは4K です。

スモールおよびミディアムモデルでは，データセグメントのサイズは次のようにして計算されます。

データセグメント ＝ グローバルデータ ＋ ヒープ ＋ スタック

ここで，ヒープのサイズは_**heaplen** を使って調整することができます。

タイニィモデルでは，コードも含めたすべてが同一のセグメントに置かれるので，データセグメントは，コードと，プログラムセグメントプレフィクス（PSP）の256バイトを含む長さに調整されます。

データセグメント ＝ 256 ＋ コード ＋ グローバルデータ ＋ ヒープ ＋ スタック

コンパクトおよびラージモデルでは，near ヒープは存在しないので，データセグメントは次のように単純に求められます。

データセグメント ＝ グローバルデータ ＋ スタック

ヒュージモデルでは，スタックは別個のセグメントにとられ，各モジュールはモジュールごとにデータセグメントを持ちます。

参照項目　_**heaplen**

timezone

機能　　ローカル時刻とGMTとの差を秒単位で保持しています。

形式　　extern long *timezone*；

宣言ファイル　　time.h

解説　　***timezone***は，時刻日付関数によって使用されます。
この変数には**tzset**関数によって計算された値が入ります。すなわち，ローカル時刻とグリニッジ標準時（GMT）との差を秒単位で表わしたものが，long型の値として代入されます。

tzname

機能　　時間帯名へのポインタの配列です。

形式　　extern char ＊ *tzname*[2]；

宣言ファイル　　time.h

解説　　グローバル変数***tzname***は，時間帯名の略称を含む文字列へのポインタの配列です。
tzname[0]は，TZ環境文字列から取り出した地方時間帯名の3文字の文字列を指しています。***tzname***[1]は，TZ環境文字列から取り出した夏時間地方時間帯名を表わす3文字の文字列を指しています。夏時間帯名が無い場合には，***tzname***[1]はヌル文字列を指します。
第2章の**tzset**関数の解説も参照してください。

_*version*

機能　DOS のバージョン番号を保持しています。

形式　extern unsigned int _*version*；

宣言ファイル　dos.h

解説　**_*version*** は MS-DOS のバージョン番号を含んでおり，メジャーバージョンが低位バイトに，マイナーバージョンが高位バイトに格納されます。MS-DOS のバージョン番号"*x.y*"の x がメジャーバージョン，*y* がマイナーバージョンです。
これらの変数は，MS-DOS のバージョン2.*x* と3.*x* の両方で走るモジュールを必要とする場合に便利です。ライブラリルーチンの中には，MS-DOS のバージョンによって動作が異なるものがいくつかあり，バージョン3.*x* の下でしか動作しないものもあります(第2章の，**_open**，**creatnew**，**ioctl** の項を参照してみてください)。

第2章

Turbo C ライブラリ

この章では，Turbo C ライブラリ中の各関数の詳細な解説を行なっています。

なお，テキストビデオライブラリ，グラフィックスライブラリ，BIOS サポートライブラリ，日本語処理ライブラリ，サウンドライブラリについては，Turbo C 標準ライブラリとは別に，それぞれ節をわけて解説してあります。

次に示したのは，この章をどのように使えばよいかを示すためのサンプルです。

関数名

機能　　そのルーチンが何を行なうかの要約です。

形式　　# include 〈header.h〉

（ヘッダファイルには，関数のプロトタイプや，その関数で使用される定数および列挙型データなどの定義が含まれています。# include 文は，その関数の呼び出しに際して必要な場合にのみ示しています）。

routine(modifier *parameter*[,...])；

（その関数の宣言の形式です。パラメータ名はイタリック（斜字体）で書かれています。[,...]は，それ以降に他のパラメータを指定してもよいことを示しています）。

プロトタイプ　　header.h

（その関数のプロトタイプを含んでいるヘッダファイル名です。いくつか

の関数のプロトタイプは，複数のヘッダファイルに含まれていますが，その場合にはそれぞれを記してあります)。

解説　ここでは，その関数が何を行なうのか，どのようなパラメータをとるのか，また，その関数および関連するルーチンを使用する際に必要となる事柄について説明しています。

戻り値　その関数が返す値がある場合にはここで示します。その関数がグローバル変数 ***errno*** をセットする場合はそれについても述べます。

可搬性　その関数を使用できるシステムおよび処理系を示します。これには，UNIX，PC-9801あるいはIBM PCおよびその互換機，ANSI Cなどが含まれます。

関連項目　その関数に関連して参照するとよいルーチンを示します。ここにあげた関数名にピリオド3つの省略記号が含まれているときは（**exec...**など)，そのルーチンはファミリィをなしていることを意味しています。

例　その関数をどのように使えばよいかを示すサンプルプログラムが載っているものもいくつかあります。

Turbo C 標準関数リファレンス

abort

機能　プロセスを異常終了させます。

形式　void abort(void)；

プロトタイプ　stdlib.h，process.h

解説　**abort** 関数は，終了メッセージ(Abnormal program termination)を *stderr* に出力し，終了コード3で_**exit** を呼び出して，プログラムを異常終了させます。

戻り値　**abort** そのものには戻り値はありませんが，**abort** によってプログラムを終了させると，親プロセスまたは DOS に3を返すことになります。

可搬性　**abort** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**assert，atexit，exit，_exit，raise，signal，spawn...**

abs

機能　整数の絶対値を返します。

形式

```
#include <math.h>
int abs(int x);
```

プロトタイプ　math.h，stdlib.h

解説　**abs** は，整数引数 x の絶対値を返します。stdlib.h がインクルードされているときは，**abs** はマクロとして扱われ，インラインコードに展開されます。**abs** をマクロではなく関数として扱いたい場合は，プログラムの中に次の1行を入れておきます。

```
#undef abs
```

これは，#include <stdlib.h>の後に書いてください。

戻り値　**abs** は，0～32767の範囲の整数を返します。ただし，引数が-32768の場合は-32768が返されます。

可搬性　**abs** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**cabs**，**fabs**，**labs**

absread

機能　ディスクセクタを直接読み込みます。

形式　int absread(int *drive*, int *nsects*, int *lsect*, void * *buffer*);

プロトタイプ　dos.h

解説　**absread** は，指定されたディスクセクタを読み込みます。これは，ディスクの論理的構造を無視し，ファイル，FAT，ディレクトリを考慮しません。**absread** は，DOS 割り込み0x25を通して，指定したディスクのセクタを読み込みます。

drive　= 読み込むドライブ番号（0=A，1=B，・・・）
nsects = 読み込むセクタ数
lsect　= 読み込みを開始する論理セクタ番号
buffer = 読み込まれるデータが格納される領域の先頭アドレス

指定されたセクタ数は，*buffer* が指すセグメント内のメモリ量によって制限を受けます。したがって，**absread** の1回の呼び出しで読み込める最大のデータ量は64K バイトということになります。

戻り値　読み込みが成功した場合，**absread** は0を返します。
エラーがあった場合は-1を返し，***errno*** にはシステムコールが返した AX レジスタの値がセットされます。***errno*** の意味については，MS-DOS のプログラマーズマニュアルを参照してください。

可搬性　**absread** は MS-DOS に特有の関数です。

関連項目　**abswrite**，**bios98disk**，**biosdisk**

abswrite

機能　ディスクセクタに直接書き込みを行ないます。

形式　int abswrite(int *drive*, int *nsects*, int *lsect*, void * *buffer*);

プロトタイプ　dos.h

解説　**abswrite** は，指定されたディスクセクタに書き込みを行ないます。これは，ディスクの論理的構造を無視し，ファイル，FAT，ディレクトリを考慮しません。

注意：適切に使用されなかった場合，**abswrite** は，ファイル，ディレクトリ，および FAT を上書きして壊してしまうことがあります。

abswrite は，DOS 割り込み0x26で，指定したディスクのセクタに書き込みを行ないます。

drive　＝ 書き込むドライブ番号（0＝A，1＝B，・・・）
nsects ＝ 書き込むセクタ数
lsect　＝ 書き込みを開始する論理セクタ番号
buffer ＝ 書き込まれるデータが格納される領域の先頭アドレス

書き込まれるたセクタ数は，*buffer* が指すセグメント内のメモリ量によって制限を受けます。したがって，**abswrite** の1回の呼び出しで書き込める最大のデータ量は64K バイトということになります。

戻り値　書き込みに成功した場合，**abswrite** は0を返します。
エラーがあった場合は-1を返し，*errno* にはシステムコールが返した AX レジスタの値がセットされます。*errno* の意味については，MS-DOS のプログラマーズマニュアルを参照してください。

可搬性　**abswrite** は MS-DOS に特有の関数です。

関連項目　　absread, bios98disk, biosdisk

access

機能　　ファイルのアクセス権を決定します。

形式　　int access(const char * *filename*, int *amode*)；

プロトタイプ　io.h

解説　　**access** は，*filename* で指定されたファイルをチェックして，そのファイルが存在するかどうか，読み込みが可能かどうか，書き込みが可能かどうか，また実行可能かどうかを調べます。
amode が示すビットパターンは，次のような意味を持ちます。

06	読み込みと書き込み許可のチェック
04	読み込み許可のチェック
02	書き込み許可のチェック
01	実行のチェック（無視されます）
00	ファイルが存在しているかどうかのチェック

注意：MS-DOS ではすべてのファイルが読み込み可能です（*amode*＝04）。したがって00と04は同じ結果になります。同様に，MS-DOS では書き込み可能ならば自動的に読み込み可能となるので，06と02は等しい意味を持つことになります。

filename がディレクトリの場合は，**access** は単にそのディレクトリが存在しているかどうかをチェックします。

戻り値　　指定されたアクセス権（*amode* に指定した値）が許されている場合は0を返します。そうでない場合は-1を返し，***errno*** に次のような値をセットします。

ENOENT	パス名あるいはファイル名が見つからなかった
EACCES	許可されていない

可搬性　access は UNIX システムで使用できます。

関連項目　chmod, fstat, stat

例

```
#include <stdio.h>
#include <io.h>

/* ファイル名が存在すれば1を返し，存在しなければ0を返す */

int file_exists(char *filename)
{
   return (access(filename, 0) == 0);
}

main()
{
   printf("Does NOTEXIST.FIL exist: %s¥n",
           file_exists("NOTEXIST.FIL") ? "YES" : "NO");
}
```

プログラム出力

```
Does NOTEXIST.FIL exist: NO
```

acos

機能 逆余弦を求めます。

形式 # include <math.h>
double acos(double *x) ;

プロトタイプ math.h

解説 **acos** は，入力値の逆余弦（アークコサイン）を返します。引数 *x* は，-1から1の間になければなりません。この範囲外の値を引数として与えると，**acos** は0を返し，***errno*** に EDOM（定義域エラー）をセットします。

戻り値 **acos** は，0〜π の値を返します。
この関数に対するエラー処理は，関数 **matherr** を使って変更することができます。

可搬性 **acos** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **asin**，**atan**，**atan2**，**cos**，**cosh**，**matherr**，**sin**，**sinh**，**tan**，**tanh**

allocmem

機能　DOSのメモリセグメントを割り当てます。

形式　int allocmem(unsigned *size*, unsigned ＊ *segp*)；

プロトタイプ　dos.h

解説　**allocmem**は，DOSのシステムコール0x48を使ってフリーメモリのブロックを確保し，そのセグメントアドレスを返します。

*size*は要求するメモリの大きさ(パラグラフ単位)，*segp*は新たに割り当てられるブロックのセグメントが格納されるワードを指すポインタです。要求量にみあうメモリが残っていない場合には，*segp*が指すワードにはアドレスは代入されません。

割り当てられたブロックは，すべてパラグラフ境界に配置されます。

戻り値　**allocmem**は，メモリブロックの確保が成功した場合には-1を返します。

allocmemでエラーがあった場合には，***_doserror***と***errno***にENOMEMをセットします。

可搬性　**allocmem**はMS-DOS特有のものです。

関連項目　**coreleft**，**freemem**，**malloc**，**setblock**

asctime

機能　日付と時刻を ASCII 文字列に変換します。

形式　# include <time.h>
char ＊ asctime(const struct tm ＊ *tblock*)；

プロトタイプ　time.h

解説　**asctime** は，＊ *tblock* 中の構造体として与えられる時刻を，**ctime** の場合と同じ形式の26文字の文字列に変換します。

```
Sun Sep 16 01:03:52 1973¥n¥0
```

戻り値　**asctime** は，日付と時刻を持つ文字列を指すポインタを返します。この文字列は静的データで，**asctime** の呼び出しが行なわれるごとに上書きされます。

可搬性　**asctime** は，すべての関数は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**ctime, difftime, ftime, gmtime, localtime, stime, time, tzset**

例

```
#include <stdio.h>
#include <time.h>

main()
{
   struct tm *tm_now;
   time_t secs_now;
   char *str_now;

   time(&secs_now);                       /* 時刻を得る(秒単位) */
   str_now = ctime(&secs_now);            /* 文字列に変換する    */

   printf("The number of seconds since Jan 1, 1970 is %ld¥n",
          secs_now);
   printf("In other words, the current time is %s¥n", str_now);

   tm_now = localtime(&secs_now);     /* 構造体に変換する */
   printf("From the structure: day %d  %02d-%02d-%02d "
         "%02d:%02d:%02d¥n",
          tm_now->tm_yday, tm_now->tm_mon,
          tm_now->tm_mday, tm_now->tm_year,
          tm_now->tm_hour, tm_now->tm_min, tm_now->tm_sec);

   str_now = asctime(tm_now);            /* 構造体から文字列へ */
   printf("Once more, the current time is %s¥n", str_now);
}
```

プログラム出力

```
The number of seconds since Jan 1, 1970 is 315594553
In other words, the current time is Tue Jan 01 12:09:13 1980
From the structure: day 0  00-01-80 12:09:13
Once more, the current time is Tue Jan 01 12:09:13 1980
```

asin

機能　逆正弦を求めます。

形式

```
#include <math.h>
double asin(double x);
```

プロトタイプ　math.h

解説　**asin** は，入力値の逆正弦（アークサイン）を返します。引数 x は，-1から1の間になければなりません。この範囲外の値を引数として与えると，**asin** は0を返し，***errno*** に EDOM（定義域エラー）をセットします。

戻り値　**asin** は，$-\pi/2 \sim \pi/2$ の値を返します。
この関数に対するエラー処理は，関数 **matherr** を使って変更することができます。

可搬性　**asin** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**acos**, **atan**, **atan2**, **cos**, **cosh**, **matherr**, **sin**, **sinh**, **tan**, **tanh**

assert

機能　　条件によってプログラムを異常終了させます。

形式

```
#include <assert.h>
#include <stdio.h>
void assert(int test);
```

プロトタイプ　assert.h

解説　　**assert** は，if 文に展開されるマクロです。引数 *test* が0（偽）の場合には，**assert** はメッセージを *stderr* に出力し（**abort** を呼び出して）プログラムを異常終了させます。
assert は，次のようなメッセージを出力します。

```
Assertion failed: <test>, file <filename>, line <linenum>
```

この中で，<test>は引数に指定した条件式，*filename* と *linenum* は，**assert** マクロが存在するソースファイル名と行番号です。
ソースコードの中で，#include <assert.h>より前に，#define NDEBUG（"デバッグ中ではない"）という指令を置いておくと，**assert** マクロはコメントとして扱われ，その部分はコンパイルされません。

戻り値　　戻り値はありません。

可搬性　　**assert** は，システム III および V を含む何種類かの UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**abort**

例

```
/* ASSERTST.C: リストに追加する項目がNULLかどうか調べる */

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

struct ITEM {
   int  key;
   int  value;
}

void additem(struct ITEM *itemptr)
{
   assert(itemptr != NULL);
   /* ... リストに項目を追加する ... */
}

main()
{
   additem(NULL);
}
```

プログラム出力

```
Assertion failed: itemptr != NULL, file C:¥TURBOC¥ASSERT.C, line 13
```

atan

機能　逆正接を求めます。

形式
```
#include <math.h>
double atan(double x);
```

プロトタイプ　math.h

解説　**atan** は，入力値の逆正接（アークタンジェント）を返します。

戻り値　**atan** は，$-\pi/2 \sim \pi/2$の値を返します。
この関数に対するエラー処理は，関数 **matherr** を使って変更することができます。

可搬性　**atan** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**acos**，**asin**，**atan2**，**cos**，**cosh**，**matherr**，**sin**，**sinh**，**tan**，**tanh**

atan2

機能 　y/x の逆正接を求めます。

形式 　# include <math.h>

double atan2(double y, double x)；

プロトタイプ 　math.h

解説 　**atan2** は，y/x の逆正接を返します。得られる角度が $\pi/2$あるいは$-\pi/2$に近い（つまり x が0に近い）場合でも正しい結果が得られるはずです。

x と y がともに0であった場合，**atan2** は ***errno*** に EDOM をセットします。

戻り値 　**atan2** は，$-\pi \sim \pi$ の値を返します。

この関数に対するエラー処理は，関数 **matherr** を使って変更することができます。

可搬性 　**atan2** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 　**acos**，**asin**，**atan**，**cos**，**cosh**，**matherr**，**sin**，**sinh**，**tan**，**tanh**

atexit

機能　終了時関数を登録します。

形式
```
#include <stdlib.h>
int atexit(atexit_t func);
```

プロトタイプ　stdlib.h

解説　**atexit** は，*func* が指している関数を"終了時関数"として登録します。プログラムが正常終了する際に，**exit** はオペレーティングシステムへ制御を戻す前に，(* *func*)()を呼び出します。呼び出される関数は *atexit_t* 型です。この型は，stdlib.h の中で **typedef** 宣言されています。
atexit が呼び出されるたびに，別の関数が登録されます。最大32個まで登録することができ，最後に登録されたものから順に実行されます。

戻り値　**atexit** は登録に成功した場合には0を返し，エラーの場合(関数を登録する領域が足りないときなど) には0でない値を返します。

可搬性　**atexit** は ANSI C と互換性があります。

関連項目　**abort**，**_exit**，**exit**，**spawn...**

例

```
#include <stdlib.h>
#include <stdio.h>

atexit_t exit_fn1(void)
{
   printf("Exit Function 1 called¥n");
}

atexit_t exit_fn2(void)
{
   printf("Exit Function 2 called¥n");
}

main()
{
   atexit(exit_fn1);     /* exit_fn1を登録 */
   atexit(exit_fn2);     /* exit_fn2を登録 */
   printf("Main quitting ...¥n");
}
```

プログラム出力

```
Main quitting ...
Exit Function 2 called
Exit Function 1 called
```

atof

機能　文字列を浮動小数点数に変換します。

形式
```
#include <math.h>
double atof(const char *s);
```

プロトタイプ　math.h, stdlib.h

解説　**atof** は，*s* が指している文字列を double 型に変換します。この関数は，浮動小数点数を表現する文字として以下のものを認識します。

- ■文字列の前後にあるタブと空白
- ■符号
- ■数字の並びと小数点（数字は小数点前後にあってもよい）
- ■指数部を示す e または E と，それに続く符号つき整数

変換される文字列は，次の一般形式に沿っていなければなりません。

```
[ws][sn][ddd][.][ddd][fmt[sn]ddd]
```

ここで，*ws* はホワイトスペース（空白およびタブ），*sn* は符号，*ddd* は数字の並び，*fmt* は e または E を示しています。
atof はさらに，+INF および-INF を正および負の無限大として，+NAN および-NAN を非数（not a number）として認識します。
atof は，認識できない文字が現われた段階で変換をやめます。

戻り値　**atof** は，入力文字列を変換した結果の値を返します。文字列が double 型の数値に変換できない場合には0を返します。
オーバーフローが起きた場合には，**atof** は正または負の HUGE_VAL を返します（**matherr** は呼ばれません）。

可搬性　**atof** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**atoi，atol，ecvt，fcvt，gcvt，strtod**

atoi

機能　文字列を整数に変換します。

形式　int atoi(const char * *s*);

プロトタイプ　stdlib.h

解説　**atoi**は，*s*が指している文字列をint型に変換します。この関数は，整数値を表現する文字として以下のものを認識します。

■数字の前後にあるタブと空白
■符号
■数字の並び

変換される文字列は，次の一般形式に沿っていなければなりません。

```
[ws][sn][ddd]
```

ここで，wsはホワイトスペース（空白およびタブ），snは符号，dddは数字の並びを示しています。
atoiは，認識できない文字が現われた段階で変換をやめます。
数値のオーバーフローについては考慮されません。

戻り値　**atoi**は，入力文字列を変換した結果の値を返します。文字列がint型の数値に変換できない場合には0を返します。

可搬性　**atoi**はUNIXシステムで使用でき，ANSI Cと互換性があります。

関連項目　**atof**，**atol**，**ecvt**，**fcvt**，**gcvt**

atol

機能　文字列を倍長整数（**long**）に変換します。

形式　long atol(const char * *s*)；

プロトタイプ　stdlib.h

解説　**atol** は，*s* が指している文字列を long 型に変換します。**atol** は，整数値を表現する文字として以下のものを認識します。

■数字の前後にあるタブと空白
■符号
■数字の並び

変換される文字列は，次の一般形式に沿っていなければなりません。

```
[ws][sn][ddd]
```

ここで，ws はホワイトスペース（空白およびタブ），sn は符号，ddd は数字の並びを示しています。
atol は，認識できない文字が現われた段階で変換をやめます。
数値のオーバーフローについては考慮されません。

戻り値　**atol** は，入力文字列を変換した結果の値を返します。文字列が long 型の数値に変換できない場合には0を返します。

可搬性　**atol** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**atof**，**atoi**，**ecvt**，**fcvt**，**gcvt**，**strtol**，**strtoul**

beep（PC-9801のみ）

機能　ビープ音を発生させます。

形式　void beep(unsigned int *time)*；

プロトタイプ　dos.h

機能説明　**beep** は，**sound** によって設定されている周期のビープ音を，*time* に指定した繰り返し回数だけ発生させます。

戻り値　戻り値はありません。

可搬性　この関数は PC-9801シリーズでのみ動作します。

関連項目　**sound**

例　**sound**(PC-9801)を参照してください。

bdos

機能 DOS システムコールを実行します。

形式 int bdos(int *dosfun*, unsigned *dosdx*, unsigned *dosal*)；

プロトタイプ dos.h

解説 **bdos** を使って，DOS システムコールの多くに直接アクセスすることができます。各システムコールの詳細については，MS-DOS プログラマーズリファレンスマニュアルを参照してください。
引数に整数を必要とするシステムコールの場合は **bdos** を使用します。
ラージデータモデル（コンパクト，ラージ，ヒュージ）では，引数としてポインタを必要とするシステムコールについては，**bdos** ではなく **bdosptr** を使う必要があるので注意してください。
引数 *dosfun* は，MS-DOS プログラマーズリファレンスマニュアルで定義されているファンクション番号です。
引数 *dosdx* は，DX レジスタの値です。
引数 *dosal* は，AL レジスタの値です。

戻り値 **bdos** の戻り値は，システムコールによって AX レジスタにセットされた値です。

可搬性 **bdos** は MS-DOS に特有の関数です。

関連項目 **bdosptr**，**geninterrupt**，**int86**，**int86x**，**intdos**，**intdosx**

例

```
#include <stdio.h>
#include <dos.h>

/* カレントドライブ ('A', 'B', ・・・) を得る */

char current_drive(void)
{
   char curdrive;

   curdrive = bdos(0x19,0,0);  /* カレントドライブ (0,1,・・・)  */
   return ( 'A' + curdrive );
}

main()
{
   printf("The current drive is %c:¥n", current_drive());
}
```

プログラム出力

```
The current drive is A:
```

bdosptr

機能　DOS システムコールを実行します。

形式　int bdosptr(int *dosfun*, void * *argument*, unsigned *dosal*)；

プロトタイプ　dos.h

解説　**bdosptr** を使って，DOS システムコールの多くに直接アクセスすることができます。各システムコールの詳細については，MS-DOS プログラマーズリファレンスマニュアルを参照してください。

ポインタ引数を必要とするシステムコールについては **bdosptr** を使用します。

ラージデータモデル（コンパクト，ラージ，ヒュージ）では，引数としてポインタを必要とするシステムコールに対しては，(**bdos** ではなく）**bdosptr** を使わなければなりません。

引数 *dosfun* は，MS-DOS プログラマーズリファレンスマニュアルで定義されているファンクション番号です。

引数 *argument* は，スモールデータモデルではシステムコールによって使用される DX の値，ラージデータモデルでは DS：DX の値を与えます。

引数 *dosal* は，AL レジスタの値です。

戻り値　**bdosptr** は，システムコールが正しく実行された場合には AX の値を返し，エラーがあった場合は ***errno*** と ***_doserrno*** がセットされます。

可搬性　**bdosptr** は MS-DOS に特有の関数です。

関連項目　**bdos，geninterrupt，int86，int86x，intdos，intdosx**

例　**harderr** を参照してください。

brk

機能　データセグメントのスペース割り当てを変更します。

形式　int brk(void * *addr*);

プロトタイプ　alloc.h

解説　**brk** は，呼び出したプログラムのデータセグメントに割り当てられているメモリ領域の大きさを，動的に変更する際に使用します。変更は，プログラムの“**ブレーク値**”を再設定することによって行ないます。ブレーク値とは，データセグメントの最後のすぐ後ろのアドレスです。割り当てられる領域は，ブレーク値の増加にしたがって増加します。
brk は，ブレーク値に ***addr*** をセットし，これに従って割り当てスペースを変更します。
システムで許される大きさを越えるスペースを確保しようとした場合にはエラーとなり，割り当てスペースの変更は行なわれません。

戻り値　割り当てスペースの変更に成功した場合，**brk** は0を返します。
変更できなかった場合には-1を返し，***errno*** を次の値にセットします。

ENOMEM　　メモリが不足した

可搬性　**brk** は **UNIX** で使用できます。

関連項目　**coreleft**，**sbrk**

bsearch

機能　配列内のバイナリサーチ（二分探索）を行ないます。

形式

```
#include <stdlib.h>
void * bsearch(const void * key, const void * base,
               size_t nelem, size_t width,
               int (* fcmp)(const void *, const void *));
```

プロトタイプ　stdlib.h

解説

bsearch は，メモリ上の *nelem* 個の要素を持つテーブル（配列）内を検索し，検索キーに一致した最初の項目のアドレスを返します。一致する項目がなかった場合は0を返します。
size_t は，unsigned int として定義されています。
テーブルの中の項目は，**bsearch** を呼び出す前に昇順にソートされていなければなりません。

- *nelem* には，テーブル中の要素の個数を与えます。
- *width* には，テーブル中の1つの要素のバイト数を指定します。

fcmp は，ユーザが定義する比較ルーチンを指すポインタです。比較ルーチンは，*elem1* と *elem2* の2つの引数をとります。2つの引数はそれぞれ比較される値を指します。比較関数は，2つの引数が指す項目（* *elem1*，* *elem2*）を比較し，その結果にしたがって整数値を返します。
* *fcmp* の戻り値は次のようになります。

- * *elem1* < 　* *elem2* の場合，0より小さい値
- * *elem1* == 　* *elem2* の場合，0
- * *elem1* > 　* *elem2* の場合，0より大きい値

通常は，*elem1* は引数 *key* で，*elem2* は検索の対象となるテーブル内の項目を指します。

fcmp が検索キーやテーブルの項目をどのように解釈するかは自由であり，ユーザが必要とする方法で定義することができます。

bsearch は，テーブル中を検索するために，*fcmp* が指すアドレスにあるルーチン（比較関数）を繰り返し呼び出すことになります。

戻り値 **bsearch** は，テーブル内で検索キーと一致した最初の項目のアドレスを返します。一致する項目がなかった場合には0を返します。

可搬性 **bsearch** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **lfind**，**lsearch**，**qsort**

例

```
#include <stdio.h>
#include <stdlib.h>

#define NELEMS(arr)    (sizeof(arr) / sizeof(arr[0]))

int numarray[] = { 123, 145, 512, 627, 800, 933 };

int numeric(int *p1, int *p2)
{
   return(*p1 - *p2);
}

/* テーブル内のキーであれば1，そうでなければ0を返す */
int lookup(int key)
{
   int *itemptr;

   /* bsearch()は，見つかった項目へのポインタを返す */
   itemptr = (int *) bsearch(&key, numarray, NELEMS(numarray),
                        sizeof(int), nomeric);
   return (itemptr != NULL);
}

main()
{
   printf("Is 512 in table? ");
   printf("%s¥n", lookup(512) ? "YES" : "NO");
}
```

プログラム出力

```
Is 512 in table? YES
```

cabs

機能　複素数の絶対値を返します。

形式　# include <math.h>
double cabs(struct complex *z*)；

プロトタイプ　math.h

解説　**cabs** は，複素数 z の絶対値を計算するマクロです。*z* は **complex** 型の構造体です。この構造体は，math.h の中で次のように定義されています。

```
struct complex {
    double x, y;
};
```

x が実数部，*y* が虚数部です。
cabs の呼び出しは，*z* の実数部と虚数部を次のようにして **sqrt** を呼び出すことと同じになります。

```
sqrt(z.x * z.x + z.y * z.y)
```

cabs を，マクロではなく関数として扱いたい場合は，プログラムに次の1行を書いてください。

```
#undef cabs
```

戻り値　**cabs** は，*z* の絶対値を double 型で返します。オーバーフローがあった場合は HUGE_VAL を返し，***errno*** に ERANGE(結果が指定の範囲を越えた)をセットします。
cabs のエラー処理は，**matherr** 関数を使って変更することができます。

可搬性　**cabs** は UNIX システムで使用できます。

関連項目　**abs, fabs, labs, matherr**

calloc

機能　メインメモリを割り当てます。

形式

```
#include <stdlib.h>
void * calloc(size_t nitems, size_t size) ;
```

プロトタイプ　stdlib.h, alloc.h

解説　**calloc** は，Cのメモリヒープへのアクセスを提供します。ヒープは，可変サイズのメモリブロックの動的割り当てに使用することができます。木構造や線形リストなど多くのデータ構造は，通常このヒープ領域に置かれます。
スモールデータモデル（タイニィ，スモール，ミディアム）では，データセグメントの終わりからプログラムスタックの先頭までをヒープとして使用することができます。ただし，スタックの先頭の直前256バイトは，アプリケーション用のスタックの拡張領域とMS-DOSが使用する領域として確保されているため除外されます。
ラージデータモデル（コンパクト，ラージ，ヒュージ）では，プログラムスタックを越えて，メモリの物理的な最終位置までヒープとして使用可能です。
calloc は，サイズ *nitems*×*size* のブロックを確保します。ブロック中の各バイトはゼロクリアされます。

戻り値　**calloc** は，新たに確保されたブロックを指すポインタを返します。メモリが不足してブロックを確保できなかった場合，あるいは *nitems* または *size* が0だった場合には NULL を返します。

可搬性　**calloc** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**farcalloc**，**free**，**malloc**，**realloc**

ceil

機能 小数点以下の切上げを行ないます。

形式
```
#include <math.h>
double ceil(double x);
```

プロトタイプ math.h

解説 **ceil** は，*x* を下まわらない最小の整数を返します。

戻り値 **ceil** は，上で述べた整数を double で返します。

可搬性 **ceil** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **farcalloc**，**free**，**malloc**，**realloc**

cgets

機能　　コンソールから文字列を読み込みます。

形式　　char * cgets(char * *str*)；

プロトタイプ　　conio.h

解説　　**cgets** は，コンソールから文字の列を読み込み，その文字列(およびその長さ）を *str* が指す場所に格納します。

cgets は，CR/LF に出会うまで，または最大文字数まで文字を読み込みます。CR/LF を読むと，**cgets** は CR/LF を¥0（ヌル文字）に置き換えてから格納します。

cgets を呼び出す前に，*str*[0]には読み込む文字列の最大長を入れておく必要があります。**cgets** の実行が終わると，*str*[1]に実際に読み込まれた文字数がセットされています。読み込まれた文字は *str*[2]から始まり，ヌル文字で終わります。したがって *str* の長さは，最低 *str*[0]＋2バイトある必要があります。

戻り値　　成功した場合，**cgets** は&*str*[2]，つまり読み込まれた文字列の先頭を指すポインタを返します。エラー戻りはありません。

可搬性　　**cgets** は PC-9801シリーズ（またはグラフィックスディスプレイアダプタを備えた IBM PC およびその互換機）でのみ動作します。

関連項目　　**fgets**，**getch**，**getche**，**gets**

例

```
#include <stdio.h>
#include <conio.h>

main()
{
   char buffer[82];
   char *p;

   buffer[0] = 80;        /* 80文字分のスペース */
   p = cgets(buffer);
   printf("¥ncgets got %d characters: ¥"%s¥"¥n", buffer[1], p);
   printf("The returned pointer is %p, buffer[2] is at %p¥n",
          p, &buffer);

   buffer[0] = 5;         /* スペースを5文字分だけにする */
   p = cgets(buffer);
   printf("¥ncgets got %d characters: ¥"%s¥"¥n", buffer[1], p);
   printf("The returned pointer is %p, buffer[2] is at %p¥n",
          p, &buffer);
}
```

プログラム出力

```
abcdefghijflm
cgets got 12 chars: "abcdefghijklm"
The returned pointer is FEF6, buffer[2] is at FEF6
abcd
cgets got 4 characters: "abcd"
The returned pointer is FEF6, buffer[2] is at FEF6
```

chdir

機能　カレントディレクトリを変更します。

形式　int chdir(const char * *path*)；

プロトタイプ　dir.h

解説　**chdir** は，*path* で指定したディレクトリを，現在の作業ディレクトリにします。*path* は実際に存在するディレクトリでなければなりません。
path の中には，次のようにドライブ名を含めることもできます。

```
chdir("a:¥¥turboc");
```

ただしこの場合，指定ドライブのカレントディレクトリが変更されるだけで，カレントドライブは変更されません。

戻り値　変更に成功すれば0を返します。そうでなければ-1を返し，***errno*** を次のようにセットします。

ENOENT　　パス名またはファイル名が見つからない

可搬性　**chdir** は UNIX システムで使用できます。

関連項目　**getcurdir**，**getcwd**，**mkdir**，**rmdir**，**system**

_chmod

機能　ファイルのアクセスモードを変更します。

形式

```
#include <dos.h>
#include <io.h>
int _chmod(const char *path, int func [, int attrib]);
```

プロトタイプ　io.h

解説　**_chmod** は，DOS のファイル属性の読み出しまたは設定を行ないます。*func* が0の場合，**_chmod** は *path* に指定したファイルの現在の DOS 属性を返します。*func* が1の場合には，属性を *attrib* で指定した値に設定します。

attrib は，以下に示すシンボリック定数（io.h の中で定義されています）のいずれかです。

FA_RDONLY	読み出し専用
FA_HIDDEN	隠しファイル
FA_SYSTEM	システムファイル

戻り値　**_chmod** は，成功した場合はファイル属性ワードを返し，そうでなければ -1を返します。

エラーの場合，***errno*** に以下の値のいずれかがセットされます。

ENOENT	パス名またはファイル名が見つからない
EACCES	アクセスは許可されていない

可搬性　**_chmod** は MS-DOS に特有なものです。

関連項目　**chmod**，**creat**

chmod

機能　ファイルのアクセスモードを変更します。

形式

```
#include <sys¥stat.h>
int chmod(const char *path, int amode);
```

プロトタイプ　io.h

解説　**chmod** は，*path* によって与えられるファイルのアクセス許可を，*amode* で指定されたマスクにセットします。*path* は，ファイル名を含む文字列を指します。
amode は，S_IWRITE，S_IREAD (sys¥stat.h の中で定義されています) のいずれか，または両方を持つことができます。

amode の値	**アクセス許可**
S_IWRITE	書き込み可
S_IREAD	読み出し可
S_IREAD	S_IWRITE	読み出し/書き込み可

戻り値　**chmod** は，ファイルアクセスモードの変更に成功した場合は0を返し，そうでなければ-1を返します。
エラーの場合，***errno*** に以下の値のいずれかがセットされます。

ENOENT	パス名またはファイル名が見つからない
EACCES	アクセスは許可されていない

可搬性　**chmod** は UNIX システムで使用できます。

関連項目　**access**，**_chmod**，**fstat**，**open**，**sopen**，**stat**

例

```
#include <stdio.h>
#include <sys¥stat.h>
#include <io.h>

void make_read_only(char *filename)
{
   int stat;
   stat = chmod(filename, S_IREAD);
   if (stat)
      printf("couldn't make %s read-only¥n", filename);
   else
      printf("made %s read-only¥n", filename);
}

main()
{
   make_read_only("NOEXIST.FIL");
   make_read_only("MYFILE.FIL");
}
```

プログラム出力

```
couldn't make NOEXIST.FIL read-only
made MYFILE.FIL read-only
```

chsize

機能 ファイルサイズを変更します。

形式 int chsize(int *handle*, long *size*);

プロトタイプ io.h

解説 **chsize**は，*handle*に対応するファイルの大きさを変更します。*size*の値と，ファイルの最初の大きさとの関係によって，ファイルの一部を切り捨てるか，大きくするかが決まります。
オープンするファイルのモードは，書き込み可能でなければなりません。
chsizeがファイルを大きくするときには，ヌル文字（¥0）を追加します。
ファイルを切り捨てる場合は，新しいファイルの終わり以降の文字は失われます。

戻り値 **chsize**は成功すれば0を返します。失敗した場合は-1を返し，***errno***に次のいずれかをセットします。

EACCESS	アクセスが拒否された
EBADF	ファイル番号が正しくない
ENOSPC	UNIXである － DOSではない

可搬性 **chsize**はMS-DOSに特有の関数です。

関連項目 **close**，**_creat**，**creat**，**fopen**

_clear87

機能　浮動小数点ステータスワードをクリアします。

形式　unsigned int _clear87(void)；

プロトタイプ　float.h

解説　**_clear87**は，浮動小数点ステータスワードをクリアします。浮動小数点ステータスワードは，8087/80287ステータスワードと，8087/80287例外ハンドラによって検出された他の条件からなるものです。

戻り値　返される値のビットパターンは，浮動小数点ステータスワードを示しています。ステータスワードの詳細については，float.hで定義されている定数を参照してください。

関連項目　**_control87**，**_fpreset**，**_status87**

例　**_control87**を参照してください。

clearerr

機能 エラー状態をリセットする

形式 #include <stdio.h>
void clearerr(FILE * *stream*);

プロトタイプ stdio.h

解説 **clearerr** は，*stream* に指定されたのエラー標識とファイル終了標識を0にします。
エラー標識がセットされると，**clearerr** または **rewind** の呼び出しが行なわれるまで，そのストリームに対する操作は常にエラーを返します。
ファイル終了標識は，入力操作によってリセットされます。

戻り値 戻り値はありません。

可搬性 **clearerr** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **eof**，**feof**，**ferror**，**perror**，**rewind**

clock

機能　プロセッサ時間を得ます。

形式　# include <time.h>

clock_t clock(void);

プロトタイプ　time.h

説明　**clock** は，2つのイベントの間の時間を測定するのに使うことができます。その時間を秒単位で表現するには，**clock** が返した値をマクロ *CLK_TCK* で割る必要があります。

戻り値　**clock** は，プログラムの開始から経過したプロセッサ時間を返します。プロセッサ時間が得られない，あるいはその値が表現不可能の場合は，関数は -1を返します。

可搬性　**clock** は ANSI C と互換性があります。

例

```
#include <time.h>
#include <stdio.h>

void main()
{
   clock_t start, end;

   start = clock();

   /* 時間を測るコードがここに入る */

   end = clock();
   printf("The time was: %f¥n", (end - start) / CLK_TCK);
}
```

_close

機能　ファイルをクローズします。

形式　int _close(int *handle*);

プロトタイプ　io.h

解説　**_close** は，*handle* に結びつけられているファイルをクローズします。*handle* は，**_creat**，**creat**，**creatnew**，**creattemp**，**dup**，**dup2**，**_open**，**open** の呼び出しによって得られたファイルハンドルを持っています。

注意：**_close** は，ファイルの最後に CTRL-Z（0x1a）を書き込みません。ファイルの最後に CTRL-Z が必要であれば，クローズする前になんらかの方法で書き込んでおかなければなりません。

戻り値　クローズに成功した場合は0を返し，そうでなければ-1を返します。*handle* が有効なオープンファイルでなかった場合にはエラーとなり，***errno*** に次の値がセットされます。

EBADF	ファイル番号が正しくない

可搬性　**_close** は MS-DOS 特有のものです。

関連項目　**close**，**_creat**，**open**，**read**，**write**

close

機能 ファイルをクローズします。

形式 int close(int *handle*);

プロトタイプ io.h

解説 **close** は, *handle* に結びつけられているファイルハンドルをクローズします。*handle* は, **_cerat**, **creat**, **creatnew**, **creattemp**, **dup**, **dup2**, **_open**, **open** の呼び出しによって得られたファイルハンドルです。

注意:**close** は, ファイルの最後に CTRL-Z(0x1a)を書き込みません。ファイルの最後に CTRL-Z が必要であれば, クローズする前になんらかの方法で書き込んでおかなければなりません。

戻り値 クローズに成功した場合は0を返し, そうでなければ-1を返します。*handle* が有効なオープンファイルでなかった場合にはエラーとなり, *errno* に次の値がセットされます。

EBADF　　ファイル番号が正しくない

可搬性 **close** は UNIX システムで使用できます。

関連項目 **chsize**, **_close**, **creat**, **creatnew**, **dup**, **fclose**, **open**, **sopen**

_control87

機能　浮動小数点制御ワードを操作します。

形式　unsigned int _control87(unsigned int *new*, unsigned int *mask*)；

プロトタイプ　float.h

解説　**_control87**は，浮動小数点制御ワードの取りだし，または変更に使用します。

浮動小数点制御ワードは，unsigned int で，ビットごとに浮動小数点パッケージのモードを示すようになっています。モードには，精度，無限値や丸めの取り扱いなどがあります。こうしたモードを変更することによって，浮動小数点の例外処理をマスクしたり，アンマスクしたりすることができます。

_control87 は，*mask* の各ビットを *new* の各ビットに対応させます。*mask* のビットが1の場合，*new* の対応するビットが，浮動小数点制御ワードの同じビットの新しい値を示すことになり，制御ワードにはこの新しい値がセットされます。

以下は，この動作の例です。

```
  元の制御ワード：  0100 0011 0110 0011

            mask:  1000 0001 0100 1111
         newvals:  1110 1001 0000 0101

変更されるビット：  1xxx xxx1 x0xx 0101
```

mask が0の場合，**_control87** は浮動小数点制御ワードを変更せずに返します。

戻り値　返される値のビットパターンは，浮動小数点制御ワードの新しい値を表わしています。**_control87** が返すビットパターンの詳細な定義については，float.h を参照してください。

関連項目　　_clear87, _fpreset, signal, _status87

例

```
#include <math.h>
#include <float.h>
#include <stdio.h>

#difine CW_NEW  (CW_DEFAULT | EM_ZERODIVIDE | EM_OVERFLOW)
#define MASK_ALL  (0xFFFF)

main()
{
   float a, b, c;

   _control87(CW_NEW|EM_INVALID, MASK_ALL);
   a = 1.0;
   b = 0.0;
   c = a/b;

   if(_status87() & SW_ZERODIVIDE)
   {
      fprintf(stderr, "DIVISION BY ZERO.¥n");
      _clear87();
      return(1);
   }
}
```

coreleft

機能 未使用メモリの量を調べて返します。

形式 ・タイニィ，スモール，ミディアムモデルの場合

```
unsigned coreleft(void);
```

・コンパクト，ラージ，ヒュージモデルの場合

```
unsigned long coreleft(void);
```

プロトタイプ alloc.h

解説 **coreleft** は，未使用のメモリ量返します。返される値は，メモリモデルがスモールデータモデルか，ラージデータモデルかによって異なります。

戻り値 ラージデータモデルでは，ヒープとスタックの間の未使用メモリの量を返します。
スモールデータモデルでは，スタックとデータセグメントの間の未使用メモリの量から256バイトを引いた値を返します。

可搬性 **coreleft** は MS-DOS に特有の関数です。

関連項目 **allocmem**，**brk**，**farcoreleft**，**malloc**

cos

機能　余弦を求めます。

形式

```
#include <math.h>
double cos(double x);
```

プロトタイプ　math.h

解説　**cos** は，入力値の余弦(サイン)を返します。角度 *x* はラジアン単位で指定します。

戻り値　**cos** は，-1から1の範囲で値を返します。
この関数に対するエラー処理は，関数 **matherr** を使って変更することができます。

可搬性　**cos** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**acos**，**asin**，**atan**，**atan2**，**cosh**，**matherr**，**sin**，**sinh**，**tan**，**tanh**

cosh

機能 双曲線余弦を求めます。

形式 # include <math.h>
double cosh(double *x)*;

プロトタイプ math.h

解説 **cosh** は，実数引数に対する双曲線余弦（ハイパボリックサイン）を計算します。

戻り値 引数の双曲線余弦を返します。
計算結果がオーバーフローしてしまう場合には，**cosh** は適切な符号をつけた HUGE_VAL を返し，***errno*** に ERANGE をセットします。
このルーチンのエラー処理は，**matherr** 関数を通じて変更することができます。

可搬性 **cosh** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **acos, asin, atan, atan2, cos, matherr, sin, sinh, tan, tanh**

country

機能　国別情報を返します。

形式　# include <dos.h>

struct country ＊ country(int xcode, struct country ＊ cp)；

プロトタイプ　dos.h

解説　**country** は，国別情報（日付，時刻，通貨など）をどのような書式で表現するかを指定します。この関数によってセットされる値は，使用している MS-DOS のバージョンによって異なります。
cp の値が-1であると，現在の国は *xcode* の値にセットされます。*xcode* は0であってはなりません。*cp* が-1以外の場合には，*cp* が指す構造体 **country** が，以下のいずれかの情報で埋められます。

■現在の国（*xcode* が0の場合）
■ *xcode* で指定された国

構造体 **country** は，dos.h の中で以下のように定義されています。

```
struct country {
   int co_date;          /* 日付書式           */
   char co_curr[5];      /* 通貨記号           */
   char co_thsep[2];     /* 3桁ごとの区切り記号 */
   char co_desep[2];     /* 10進数分離記号     */
   char co_dtsep[2];     /* 日付分離記号       */
   char co_tmsep[2];     /* 時刻分離記号       */
   char co_currstyle;    /* 通貨の形式         */
   char co_digits;       /* 通貨の桁数         */
   char co_time;         /* 時刻形式           */
   long co_case;         /* ケースマップ       */
   char co_dasep;        /* データの区切り記号 */
   char co_fill[10];     /* フィラ(埋める文字) */
};
```

co_date の日付書式は，次のようになります。

0　米国式（月-日-年）
1　欧州式（日-月-年）
2　日本式（年-月-日）

co_currstyle が示す通貨表示形式は，以下のようになります。

0　通貨記号は数値の前，通貨記号と数値の間には空白なし
1　通貨記号は数値の後ろ，通貨記号と数値の間には空白なし
2　通貨記号は数値の前，通貨記号と数値の間に空白1個
3　通貨記号は数値の後ろ，通貨記号と数値の間に空白1個

戻り値　成功すれば，**country** はポインタ引数 *cp* を返します。エラーがあった場合には NULL を返します。

可搬性　**country** は MS-DOS 3.0以降で使用できます。

cprintf

機能　画面に書式つき出力を行ないます。

形式　int cprintf(const char * *format* [, *argument*,...]);

プロトタイプ　conio.h

解説　**cprintf** は，*format* によって指される書式文字列中の書式指定を，*format* の後に続く各引数に適用し，書式化されたデータを画面上のカレントテキストウィンドウに直接出力します。書式指定は，後に続く引数と同じ数だけなければなりません。

書式指定に関する詳細な情報については，**printf** の解説を参照してください。

cprintf は，**printf** および **fprintf** とは異なり，改行文字（¥n）を復帰改行ペア（¥r¥n）に変換しません。

戻り値　**cprintf** は，出力した文字数を返します。

可搬性　**cprintf** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。

関連項目　***directvideo***(変数)，**fprintf**，**printf**，**putch**，**sprintf**，**vprintf**

例　**printf** を参照してください。

cputs

機能　画面に文字列を出力します。

形式　int cputs(const char * *str*)；

プロトタイプ　conio.h

解説　**cputs** は，ヌルで終わる文字列 *str* を，カレントテキストウインドウに書きます。復改文字は付加されません。
PC-9801では，文字列は直接テキスト VRAM に書き込まれます。IBM PC の場合には，グローバル変数 ***directvideo*** の値によって，画面メモリに直接書き込まれるか，BIOS を通じて出力されるかが決まります。
cputs は，**puts** とは異なり，改行文字（¥n）を復帰改行ペア（¥r¥n）に変換しません。

戻り値　**cputs** は，最後に書いた文字を返します。

可搬性　**cputs** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。

関連項目　***directvideo***(変数)，**putch**，**puts**

_creat

機能　新しいファイルを作成する，あるいは既存のファイルを書きなおします。

形式　# include <dos.h>

int _creat(const char * *path*, int *attrib*) ;

プロトタイプ　io.h

解説　**_creat** は，新しいファイルを作成するか，あるいはすでに存在するファイルを書きなおすための準備を行ないます。作成または書きなおされるファイルは，*path* によって与えられます。

_creat は，DOS の属性ワード *attrib* を引数にとります。**_creat** の呼び出しでは，どの属性ビットもセットすることができます。ファイルは，常にバイナリモードでオープンされます。ファイルの作成に成功すれば，ファイルポインタはファイルの先頭にセットされます。ファイルは，読み込み/書き込み可でオープンされます。

ファイルがすでに存在している場合，そのサイズは0にリセットされます（これは，そのファイルを削除してから同じ名前で新たに作成するのと，基本的には同じことです)。

_creat に渡す引数 *attrib* は，(dos.h で定義されている) 以下のいずれかの値をとることができます。

FA_RDONLY	読み出し専用属性
FA_HIDDEN	隠しファイル
FA_SYSTEM	システムファイル

戻り値　動作に成功した場合，**_creat** は新しいファイルハンドル（負でない整数）を返し，エラーの場合には-1を返します。
エラーがあった場合，***errno*** には次の値のいずれかがセットされます。

ENOENT	パス名またはファイル名が見つからない
EMFILE	オープンファイルが多すぎる
EACCES	アクセスは許可されていない

可搬性　**_creat** は MS-DOS に特有の関数です。

関連項目　**_chmod**，**chsize**，**_close**，**close**，**creat**，**creatnew**，**creattemp**

creat

機能　新しいファイルを作成する，あるいは既存のファイルを書きなおします。

形式　#include <sys¥stat.h>

int creat(const char * *path*, int *amode*);

プロトタイプ　io.h

解説　**creat** は，新しいファイルを作成するか，あるいはすでに存在するファイルを書きなおすために使用します。ファイル名は，*path* が指す文字列です。*amode* は，ファイルを新たに作成するときにのみ用いられます。

creat によって作成されるファイルは，常にグローバル変数 ***_fmode*** によって指定される変換モード（O_TEXT または O_BINARY）で作成されます。

ファイルがすでに存在していて，書き込み属性がセットされている場合には，**creat** はそのファイルのサイズを0にして，属性はそのまま変更しません。すでに存在しているファイルが読み出し専用属性であると，**creat** の呼び出しはエラーとなり，そのファイルは変更されません。

creat の呼び出しでは，アクセスモードワード *amode* 内の S_IWRITE ビットのみが調べられます。このビットが1であれば，そのファイルは書き込みが可能です。このビットが0であれば読み出し専用になります。これ以外の DOS の属性は，すべて0にセットされます。

amode には，以下の値のいずれかを指定します(sys¥stat.h の中で定義されています)。

amode の値	アクセス許可
S_IWRITE	書き込み可
S_IREAD	読み出し可
S_IREAD ¦ S_IWRITE	読み出し/書き込み可

注意：MS-DOSでは，書き込み可であれば同時に読み出しも可になります。

戻り値　動作に成功した場合には，新しいファイルハンドル（負でない整数）が返され，エラーの場合には-1が返されます。
エラーがあった場合，***errno***には次のいずれかの値がセットされます。

ENOENT	パス名またはファイル名が見つからない
EMFILE	オープンされているファイルが多すぎる
EACCES	アクセスは許可されていない

可搬性　**creat**はUNIXシステムで使用できます。

関連項目　**chmod**，**chsize**，**close**，**_create**，**creatnew**，**creattemp**，**dup**，**dup2**，***_fmode***(変数)，**fopen**，**open**，**sopen**，**write**

creatnew

機能　新しいファイルを作成します。

形式

```
#include <dos.h>
int creatnew(const char * path, int attrib);
```

プロトタイプ　io.h

解説　**reatnew** は，**_creat** とほとんど同じです。異なる点は，指定したファイルがすでに存在している場合には **creatnew** の呼び出しはエラーとなり，そのファイルは変化を受けないということです。
creatnew の引数 *attrib* には，(dos.h の中で定義されている)以下に示す定数のいずれかを指定します。

FA_RDONLY	読み出し専用
FA_HIDDEN	隠しファイル
FA_SYSTEM	システムファイル

戻り値　動作に成功した場合には，新しいファイルハンドル（負でない整数）が返され，エラーの場合には-1が返されます。
エラーがあった場合，***errno*** には次のいずれかの値がセットされます。

EEXIT	すでにファイルが存在する
ENOENT	パス名またはファイル名が見つからない
EMFILE	オープンされているファイルが多すぎる
EACCES	アクセスは許可されていない

可搬性　**creatnew** は MS-DOS 3.0 に特有の関数で，それより前のバージョンの DOS では動作しません。

関連項目　**close**，**_creat**，**creat**，**creattemp**，**dup**，***_fmode***(変数)，**open**

creattemp

機能 指定されたディレクトリ中にユニークなファイルを作成します。

形式 # include <dos.h>

int creattemp(char * *path*, int *attrib*) ;

プロトタイプ io.h

解説 **creattemp**によって作成されるファイルは，常にグローバル変数 ***_fmode*** で指定されている変換モード（O_TEXT または O_BINARY）で作成されます。

引数 *path* はパス名で，最後に円記号（¥）をつけます。*path* に指定されたディレクトリの中でユニークな（存在するどのファイルとも一致しない）適当なファイル名がつけられます。作成されたファイル名は *path* が指す場所におさめられます。*path* は，結果となるファイル名を格納するのに充分な大きさがなければなりません。作成されたファイルは，プログラムが終了するときに，自動的に削除されるということはありません。

creattemp は，DOS の属性ワード *attrib* を引数にとります。**_creat** の呼び出しでは，どの属性ビットもセットすることができます。ファイルは，常にバイナリモードでオープンされます。ファイルの作成に成功すれば，ファイルポインタはファイルの先頭にセットされます。ファイルは，読み込み/書き込み可でオープンされます。

attrib には，以下に示す定数のいずれかを指定します(これらは dos.h の中で定義されています)。

FA_RDONLY	読み出し専用
FA_HIDDEN	隠しファイル
FA_SYSTEM	システムファイル

戻り値 動作に成功した場合には，新しいファイルハンドル（負でない整数）が返され，エラーの場合には-1が返されます。
エラーがあった場合，***errno*** には次のいずれかの値がセットされます。

ENOENT	パス名またはファイル名が見つからない
EMFILE	オープンされているファイルが多すぎる
EACCES	アクセスは許可されていない

可搬性 **creattemp** は MS-DOS 3.0 に特有の関数で，それより前のバージョンの DOS では動作しません。

関連項目 **close**，**_creat**，**creat**，**creatnew**，**dup**，***_fmode***(変数)，**open**

cscanf

機能　コンソールから書式つき入力を行ないます。

形式　int cscanf(char * *format* [, *address*,...]) ;

プロトタイプ　conio.h

解説　**cscanf** は，一連の入力フィールドをスキャンして，一度に1文字ずつコンソールから直接文字を読み込みます。次に，引数 *format* によって指される書式文字列中の書式指定にしたがって，各フィールドを書式化します。最後に，*format* の後に続く各引数が示しているアドレスに，書式化した入力を格納していきます。入力は画面に直接エコーされます。書式文字列中の書式指定の個数は，その後に続くアドレスの数と同じでなければなりません。書式指定も含めて，詳細な情報については **scanf** の解説を参照してください。

cscanf は，通常のフィールドの終了（ホワイトスペース）に達する前に，フィールドのスキャンをやめる場合があります。また，いくつかの理由から入力を終了してしまうこともあります。こうした問題については **scanf** の解説を参照してください。

戻り値　**cscanf** は，正しくスキャンし，変換し，格納できた入力フィールドの数を返します。戻り値には，値が格納されなかった入力フィールドの数は含まれません。値が格納されたフィールドがなかった場合は，戻り値は0になります。

cscanf がファイルエンドを読み込もうとした場合，戻り値は EOF になります。

可搬性　**cscanf** は UNIX システムで使用でき，K&R で定義されています。

関連項目　**fscanf**, **getche**, **scanf**, **sscanf**

ctime

機能 日付と時刻を文字列に変換します。

形式 #include <time.h>

char * ctime(const time_t * *time*);

プロトタイプ time.h

解説 **ctime** は，*time* が指している時刻値(**time** 関数などによって返されたもの)を，次のような最後に改行とヌル文字のついた26文字の文字列に変換します。

```
Mon Nov 21 11:31:54 1983¥n¥0
```

各フィールドの幅は固定です。

long 型のグローバル変数 ***timezone*** は，GMT とその地方の標準時の差を秒単位で保持しています（PST－太平洋標準時の場合には8×60×60)。グローバル変数 ***daylight*** は，米国の夏時間の期間中にのみ0以外の値をとるようになっています。

戻り値 **ctime** は，日付と時刻を持つ文字列を指すポインタを返します。戻り値は静的データを指しており，これは呼び出しが行なわれるごとに上書きされます。

可搬性 **ctime** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **asctime**，***daylight***(変数)，**difftime**，**ftime**，**getdate**，**gmtime**，**localtime**，**settime**，**time**，***timezone***(変数)，**tzset**

例 **asctime** を参照してください。

ctrlbrk

機能　コントロールブレークハンドラをセットします。

形式　void ctrlbrk(int (* *handler*)(void));

プロトタイプ　dos.h

解説　**ctrlbrk** は，*handler* が指す関数を，新たなコントロールブレークハンドラとしてセットします。割り込みベクタ0x23は，この関数を呼び出すように変更されます。

ctrlbrk は，この関数を呼び出す DOS の割り込みハンドラを作りあげます。この関数が直接呼び出されることはありません。

ハンドラ関数は，どんな操作やシステムコールを行なってもかまいません。

ハンドラは，必らずしも元の位置に戻る必要はなく，**longjmp** を使ってプログラム内の任意の場所に戻ることができます。ハンドラ関数が0を返すとプログラムは異常終了します。0以外の値を返した場合には，プログラムの実行が再開されます。

戻り値　**ctrlbrk** は何も返しません。

可搬性　**ctrlbrk** は MS-DOS に特有の関数です。

関連項目　**getcbrk**，**signal**

例

```
#include <stdio.h>
#include <dos.h>

#define ABORT 0
int c_break(void);
{
   printf("Control-Break hit.  Program aborting ...¥n");
   return(ABORT);
}

main()
{
   ctrlbrk(c_break);
   for (;;) {          /* 無限ループ */
      printf("Looping ...¥n");
   }
}
```

プログラム出力

```
Looping ...
Looping ...
Looping ...
^C
Control-Break hit.  Program aborting ...
```

delay

名前　実行を一時停止させます。

形式　void delay(unsigned *milliseconds*);

プロトタイプ　dos.h

機能説明　**delay** を呼び出すと，現在実行中のプログラムは引数 *milliseconds* で指定したミリ秒数だけ実行を一時停止します。正確な停止時間は，動作環境が異なると多少変化することがあります。

戻り値　ありません。

可搬性　この関数は PC-9801（または IBM PC およびその互換機）でのみ動きます。

関連項目　**nosound**，**sleep**，**sound**

difftime

機能　2つの時刻の差を計算します。

形式

```
#include <time.h>
double difftime(time_t time2, time_t time1);
```

プロトタイプ　time.h

解説　**difftime** は，*time1* から *time2* までの経過時間を秒単位で計算します。long 型のグローバル変数 ***timezone*** は，GMT とその地方の標準時の差を秒単位で保持しています（PST－太平洋標準時の場合には8×60×60）。グローバル変数 ***daylight*** は，米国夏時間の期間中にのみ0以外の値をとるようになっています。

戻り値　**difftime** は計算の結果を double で返します。

可搬性　**difftime** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**asctime**，**ctime**，***daylight***（変数），**time**，***timezone***（変数）

disable

機能　割り込みを不可にします。

形式　# include <dos.h>
void disable(void)；

プロトタイプ　dos.h

解説　**disable** は，ハードウェア割り込みの制御が柔軟に行なえるように用意されています。
disable マクロは割り込みを不可にします。どんな外部デバイスからでも，NMI 割り込み（non-maskable interrupt）のみが許されることになります。

戻り値　戻り値はありません。

可搬性　**disable** マクロは80x86アーキテクチャに特有の関数です。

関連項目　**enable**，**getvect**

div

機能　2つの整数の除算を行なって，商と剰余を返します。

形式　#include <stdlib.h>
div_t div(int *numer*, int *denom*)；

プロトタイプ　stdlib.h

解説　**div** は2つの整数の除算を行なって，商と剰余を **div_t** 型として返します。*numer* と *denom* は，それぞれ非除数と除数です。**div_t** 型は整数の構造体で，stdlib.h の中で次のように **typedef** 宣言されています。

```
typedef struct {
    int  quot;          /* 商   */
    int  rem;           /* 剰余 */
} div_t;
```

戻り値　**div** は，*quot*（商）と *rem*（剰余）を要素に持つ構造体を返します。

可搬性　**div** は ANSI C と互換性があります。

関連項目　**ldiv**

例

```
#include <stdlib.h>

div_t x;

main()
{
   x = div(10,3);
   printf("10 div 3 = %d remainder %d¥n", x.quot, x.rem) ;
}
```

プログラム出力

```
10 div 3 = 3 remainder 1
```

dosexterr

機能　DOS の拡張エラー情報を得ます。

形式　# include <dos.h>

int dosexterr(struct DOSERROR * *eblkp*);

プロトタイプ　dos.h

解説　この関数は，DOS のシステムコールが不成功に終わった場合に，拡張エラー情報を，*eblkp* が指す **DOSERROR** 構造体に埋め込みます。この構造体は次のように定義されています。

```
struct DOSERROR {
    int exterror;        /* 拡張エラー    */
    char class;          /* エラークラス  */
    char action;         /* 動作          */
    char locus;          /* エラーの位置  */
}
```

この構造体の各値は，DOS のシステムコール0x59を通じて取得されます。*exterror* の値が0であれば，その前に行なわれた DOS コールはエラーにならなかったことを意味します。

戻り値　**dosexterr** は，**DOSERROR** 構造体中の *exterror* の値を返します。

可搬性　**dosexterr** は MS-DOS 3.0に特有の関数で，それ以前のバージョンでは動作しません。

dostounix

機能　日付と時刻を UNIX の時刻書式に変換します。

形式　#include <dos.h>

long dostounix(struct date * *d*, struct time * *t*);

プロトタイプ　dos.h

解説　**dostounix** は，**getdate** と **gettime** で得た日付と時刻を UNIX の書式に変換します。*d* は **date** 構造体を，*t* は **time** 構造体を指しており，2つの構造体はそれぞれ有効な DOS の日付および時刻に関する情報を含んでいるとします。

戻り値　**dostounix** は UNIX 形式の現在時刻，つまり1970年1月1日午前0時(GMT)からの経過時間を秒数で表わしたものが返されます。

可搬性　**dostounix** は MS-DOS 特有の関数です。

関連項目　**unixtodos**

dup

機能　ファイルハンドルを複製します。

形式　int dup(int *handle*)；

プロトタイプ　io.h

解説　**dup** は，元のファイルハンドルと次のような共通点を持つ新しいファイルハンドルを返します。

- ■同じオープンファイルあるいはデバイス
- ■同じファイルポインタ（つまり，一方のファイルポインタを変更すると，もう一方も変更されます）
- ■同じアクセスモード（読み出し，書き込み，読み出し/書き込み）

引数 *handle* は，**_creat**，**creat**，**_open**，**open**，**dup**，**dup2** の呼び出しで得られたファイルハンドルです。

戻り値　成功した場合，**dup** は新しいファイルハンドル(負でない整数)を返し，エラーの場合は-1を返します。
エラーの場合，***errno*** は次のいずれかにセットされます。

EMFILE	オープンファイルが多すぎる
EBADF	ファイル番号が正しくない

可搬性　**dup** はすべての UNIX システムで使用できます。

関連項目　**_close**，**close**，**_creat**，**creat**，**creatnew**，**creattemp**，**dup2**，**fopen**，**_open**，**open**

dup2

機能　ファイルハンドルを既存のファイルハンドルに複製します。

形式　int dup2(int *oldhandle*, int *newhandle*)；

プロトタイプ　io.h

解説　**dup2** は，元のファイルハンドルと次のような共通点を持つ新しいファイルハンドルを作成します。

- ■同じオープンファイルあるいはデバイス
- ■同じファイルポインタ（つまり，一方のファイルポインタを変更すると，もう一方も変更されます）
- ■同じアクセスモード（読み出し，書き込み，読み出し/書き込み）

dup2 は，*newhandle* の値をもつ新しいファイルハンドルを作成します。**dup2** が呼び出されたときに，*newhandle* に結びつけられたファイルがオープンされている場合は，そのファイルはクローズされます。
newhandle と *oldhandle* は，**creat**，**open**，**dup**，**dup2** の呼び出しで得られたファイルハンドルです。

戻り値　**dup2** は成功した場合は0を返し，エラーの場合は-1を返します。
エラーの場合，***errno*** は次のいずれかにセットされます。

EMFILE	オープンファイルが多すぎる
EBADF	ファイル番号が正しくない

可搬性　**dup2** はいくつかの UNIX システムで使用できます。ただし，System III では使用できません。

関連項目　**_close**，**close**，**_creat**，**creat**，**creatnew**，**creattemp**，**dup**，**fopen**，**_open**，**open**

ecvt

機能　浮動小数点数を文字列に変換します。

形式　char * ecvt(double *value*, int *ndig*, int * *dec*, int * *sign*) ;

プロトタイプ　stdlib.h

解説　**ecvt** は，*value* の値を，一番左の有効桁から始まりヌルで終わる *ndig* 桁の文字列に変換し，その文字列を指すポインタを返します。小数点が文字列の先頭から何桁目にあるかは，*dec* を通して間接的に格納されます(*dec* が負の場合は，返された最初の数字の左に小数点があることを意味します)。小数点は文字列そのものには含まれません。結果の符号が負のときは，*sign* が指すワードは0でなくなり，それ以外のときは0となります。下位の桁は四捨五入されます。

戻り値　**ecvt** の戻り値は静的データを指しているので，**ecvt** が呼ばれるたびに，その内容は上書きされます。

可搬性　**ecvt** は UNIX システムで使用できます。

関連項目　**atof**，**atoi**，**atol**，**fcvt**，**gcvt**，**printf**

__emit__

機能　リテラル値を直接コードに埋め込みます。

構文　void　__emit__(*argument*, ...)；

プロトタイプ　dos.h

説明

__emit__はインライン関数であり，コンパイル時にオブジェクトコードの中に，リテラル値を直接埋め込むために使われます。これによって，インラインアセンブリ言語やアセンブラを使わなくても，マシン語命令を生成することができます。インラインアセンブリが使えない統合開発環境の中でもこの関数を使用できます。
一般的に，**__emit__**の呼び出しでは1バイトマシンコード命令を指定しますが，この関数ではCの変数の参照などのより複雑な命令も使用できる機能をもっています。

警告!この関数は，80x86プロセッサファミリの機械語によく通じているプログラマのみが使用すべきものです。この関数を使えば，関数の中に好きなバイトを埋め込むことができますが，もし間違いがあればプログラムはおかしな動作を示し，暴走してしまいます。Turbo Cは，この関数の呼び出しが正しいかどうかはまったくチェックしません。レジスタやメモリの内容を変更する命令を埋め込んだ場合，Turbo Cはそれを認識できず，インラインアセンブリ言語において多くの場合になされるレジスタの保存も行なわれません(たとえば，インライン命令では，SIレジスタおよびDIレジスタが使われていることが認識されます)。この関数を使う場合はすべてプログラマの責任となります。

__emit__には少なくとも1個は引数を渡す必要がありますが，それ以上であればいくつ渡してもかまいません。これらの引数は，他の関数呼び出しにおける引数とは違う扱いを受けます。つまり，**__emit__**に渡される引数は変換されることはありません。
__emit__への引数の形式に対しては特別な制限があります。静的なオブ

ジェクトを初期化するのに使用できる式の形式をとっていなければなりません。つまり，整数や浮動小数点の定数，静的なオブジェクトのアドレスを使うことができます。こうした式の値は，オブジェクトコード中の呼び出しの位置に，初期化データとして使用されるのとまったく同じように埋め込まれます。自動変数や引数変数のアドレス，これらに定数オフセットをプラスまたはマイナスしたものも使用することができます。これらの引数に対しては，BPからの変数のオフセットが格納されます。
オブジェクトコードの中で占めるバイト数は，次の場合を除いて，引数の型によって決まります。

- ■その値が0から255の範囲の符号つき整定数（たとえば0x90）は，文字と同様に扱われ，1バイトを占めます。
- ■自動変数や引数変数のアドレスが使用される場合は，BPからの変数のオフセットが-128から127の範囲に入っていれば1バイト，それ以外は1ワードを占めます。

1バイトは次のようにして書くことができます。

```
__emit__(0x90);
```

値は255以下でもワードとして書きたい場合は，次のようにキャストを使うか，

```
__emit__(0xB8, (unsigned)17);
```

あるいは次のようにします。

```
__emit__(0xB8, 17u);
```

2バイトあるいは4バイトのアドレスは，アドレスをvoid near *あるいはvoid far *にキャストすることによって得られます。

戻り値　ありません。

可搬性　__emit__は，インテル80x86アーキテクチャに特有の関数です。

enable

機能　割り込みを可能にします。

形式

```
#include <dos.h>
void enable(void);
```

プロトタイプ　dos.h

解説　**enable**は，ハードウェア割り込みの制御が柔軟に行なえるように用意されています。

enableマクロは，割り込みを可能にします。これによってどのデバイスからでも割り込みができるようになります。

戻り値　戻り値はありません。

可搬性　**enable**は80x86アーキテクチャに特有の関数です。

関連項目　**disable**，**getvect**

eof

機能　ファイルエンドかどうかを調べます。

形式　int eof(int *handle*)；

プロトタイプ　io.h

解説　**eof** は，*handle* に結びつけられたファイルがファイルエンドに到達しているかどうかを決定します。

戻り値　**eof** は，現在位置がファイルの終わりであれば1を返し，そうでなければ0を返します。戻り値が-1の場合はエラーを意味し，***errno*** に次の値がセットされます。

EBADF　　ファイル番号が正しくない

関連項目　**clearerr**，**feof**，**ferror**，**perror**

exec...

機能　他のプログラムをロードして実行します。

形式

```
int execl(char * path, char * arg0, * arg1, ..., * argn, NULL) ;
int execle(char * path, char * arg0, * arg1, ..., * argn, NULL,
          char ** env) ;
int execlp(char * path, char * arg0, * arg1, ..., * argn, NULL) ;
int execlpe(char * path, char * arg0, * arg1, ..., * argn, NULL,
          char ** env) ;
int execv(char * path, char * argv[]) ;
int execve(char * path, char * argv[], char ** env) ;
int execvp(char * path, char * argv[]) ;
int execvpe(char * path, char * argv[], char ** env) ;
```

プロトタイプ　process.h

解説　**exec...**ファミリィの関数は，他のプログラムをロードし実行します。これは“子プロセス”として知られています。**exec...**の呼出しが成功すると，子プロセスは“親プロセス”にオーバーレイします。子プロセスをロードして実行するために充分なメモリ領域が存在しなければなりません。

path は，呼び出される子プロセスのファイル名です。**exec...**は，*path* を次のような MS-DOS 標準の手順で探します。

- ■拡張子もピリオドもない場合は，まず与えられたファイル名を探します。存在しなければ.COM を付加して再び探します。それも存在しなければ.EXE を付加してもう一度探します。
- ■拡張子が与えられた場合は，与えられたとおりのファイル名を探します。
- ■拡張子なしでピリオドがある場合は，拡張子なしでファイル名を探します。

exec...につけられた接尾辞，**l**，**v**，**p**，**e** は，次のような意味をもっていま

す。

p ファイル名を探す際に，DOSの環境変数PATHにセットされているすべてのディレクトリを探索します（**p**がついていない場合は，カレントディレクトリしか探索されません）。引数*path*にはっきりとディレクトリが指定されていない場合には，最初にカレントディレクトリを探し，次にルートディレクトリを探します。

l 引数ポインタ（*arg0*, *arg1*, ..., *argn*）は，別々の引数として渡されることを意味します。一般に，引数の個数があらかじめわかっているときに接尾辞**l**がついたものが使われます。

v 引数ポインタ（*argv*[0]，. . .，*argv*[*n*]）は，ポインタの配列として渡されることを意味します。一般に，渡される引数の個数が可変のときに接尾辞vがついたものが使われます。

e 引数*env*が子プロセスに渡されることを意味します。これによって子プロセスの環境を変えることができます。**e**がつかない場合，子プロセスは親プロセスの環境を受けつぎます。

exec...ファミリィの各関数は，引数に関する指定を行なう2つの接尾辞（**l**または**v**）のいずれかを含んでいなければなりません。パスのサーチ手順と環境の受けつぎ方に関する接尾辞（**p**と**e**）はオプションで，なくてもかまいません。
たとえば次のようになります。

- **execl**は，別々の引数を受け取り，ルートと現在の作業ディレクトリのみをサーチし，子プロセスは親プロセスの環境を受け継ぐ。
- **execvpe**は，引数ポインタの配列を受け取り，PATHで指定されたディレクトリをサーチし，子プロセスの環境を変えるために*env*を受け取る。

exec...関数は，子プロセスへ渡す引数を少なくとも1個（*arg0*あるいは*argv*[0]）持たなくてはなりません。この引数は慣習で普通は*path*のコピーです（別の値を使ってもエラーにはなりませんが）。MS-DOS 3.*x*では，子プロセスで*path*を使用することができますが，それより前のバージョンでは0番目の引数（*arg0*または*argv*[0]）の渡された値を使うことはできません。

接尾辞 l が使われている場合，*arg0* は通常 *path* を指し，*arg1,...,argn* は新しい引数リストを形成する文字列を指します。*argn* の次の引数 NULL は必らず必要で，リストの終わりを示します。
接尾辞 e が使われている場合，新しい環境変数のリストを引数 *env* で渡すことができます。*env* は char ポインタの配列で，各要素はヌルで終わる次のような文字列を指しています。

```
envvar = value
```

envvar は環境変数の名前であり，*value* は *envvar* にセットされる文字列です。*env* の最後の要素には NULL が入ります。*env* が NULL の場合，子プロセスは親プロセスの環境設定を受けつぎます。
arg0＋*arg1*＋. . . ＋*argn*（あるいは *argv*[0]＋*argv*[1]＋. . . ＋*argv*[*n*]）の長さは，区切りの空白も含めて128バイトより短くなくてはなりません。ヌル文字はこれには含めません。**exec...**関数が呼び出されたとき，オープンされているファイルはそのままになります。

戻り値　成功した場合，**exec...**関数は値を返しません。エラーの場合は-1を返し，***errno*** には次のいずれかがセットされます。

E2BIG	引数リストが長すぎる
EACCES	アクセス許可が否定されている
EMFILE	オープンされているファイルが多すぎる
ENOENT	パス名あるいはファイル名が見つからない
ENOEXEC	exec 書式エラー
ENOMEM	メモリが不足した

可搬性　**exec...**は MS-DOS 特有の関数です。

関連項目　**abort**，**atexit**，**_exit**，**exit**，**_fpreset**，**searchpath**，**spawn...**，**system**

例

```
#include <stdio.h>
#include <process.h>

main()
{
   int stat;

   printf("About to exec child with arg1 arg2 ...¥n");
   stat = execl("CHILD.EXE", "CHILD.EXE", "arg1", "arg2", NULL);

   /* execl は CHILD を実行できなかった場合にのみ制御を戻す */
   printf("execl("execl error = %d¥n", stat);
   exit(1);
}

/* CHILD.C */
#include <stdio.h>

main(int argc, char *argv[1]
{
   int i;

   printf("Child running ...¥n");
   for (i=0; i<argc; i++)       /* 引数を表示 */
      printf("argv[%d]: %s¥n", i, argv[1]);
}
```

プログラム出力

```
About to exec child with arg1 arg2 ...
Child running ...
argv[0]: CHILD.EXE
argv[1]: arg1
argv[2]: arg2
```

_exit

機能　　プログラムを終了させます。

形式　　void _exit(int *status*);

プロトタイプ　　process.h, stdlib.h

解説　　**_exit** は，ファイルのクローズ，出力のフラッシュ，終了時関数の呼び出しをいっさい行なわずにプロセスを終了させます。
引数 *status* は，プロセスの終了ステータスとして呼び出したプロセスに供給されます。一般的には，0は正常終了を示し，0以外の値はエラーを示すために使われます。

戻り値　　戻り値はありません。

可搬性　　**_exit** は UNIX システムで使用できます。

関連項目　　**abort**, **atexit**, **exec...**, **exit**, **spawn...**

exit

機能　プログラムを終了させます。

形式　void exit(int *status)；

プロトタイプ　prosess.h，stdlib.h

解説　**exit** は，この関数を呼び出したプロセスを終了させます。終了する前に，すべてのオープンされているファイルはクローズされ，バッファの中の出力（出力されるのを待っている）は書き出され，登録されている"終了時関数"（**atexit** を参照）が呼び出されます。
引数 *status* は，プロセスの終了ステータスとして呼び出したプロセスに供給されます。一般的には，0は正常終了を示し，0以外の値はエラーを示すために使われます。

戻り値　戻り値はありません。

可搬性　**exit** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**abort**，**atexit**，**exec...**，**_exit**，**keep**，**signal**，**spawn...**

exp

機能　指数関数 e の x 乗を返します。

形式

```
# include <math.h>
double exp(double x);
```

プロトタイプ　math.h

解説　**exp** は，指数関数 e^x を計算します。

戻り値　**exp** は，e^x を返します。

引数の値によっては，オーバーフローが起こったり，計算できないことがあります。オーバーフローが起きた場合, **exp** と **pow** は HUGE_VAL を返します。大きさが非常に大きい結果は ***errno*** に次の値をセットします。

ERANGE　　結果が指定の範囲を越えた

オーバーフローの場合には **exp** は0.0を返し，***errno*** は変更されません。
exp のエラー処理は，**matherr** 関数を使って変更することができます。

可搬性　**exp** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**frexp, ldexp, log, log10, matherr, pow, pow10, sqrt**

fabs

機能　浮動小数点数の絶対値を返します。

形式　#include <math.h>

```
double fabs(double x);
```

プロトタイプ　math.h

解説　**fabs** は，double 型の引数 x の絶対値を計算します。

戻り値　**fabs** は，*x* の絶対値を返します。エラーの戻り値はありません。

可搬性　**fabs** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**abs**，**cabs**，**labs**

farcalloc

機能　ヒープからメモリを割り当てる

形式　void far * farcalloc(unsigned long *nunits*, unsigned long *unitsz*);

プロトタイプ　alloc.c

解説　**farcalloc** は，far ヒープから各要素が *unitsz* バイトで，要素の数が *nunits* の配列用のメモリを割り当てます。
far ヒープから割り当てに関しては，次の点に注意してください。

- ■利用可能な RAM はすべて割り当てることができる。
- ■64K バイトを超えるブロックを割り当てることができる。
- ■割り当てられたブロックをアクセスするには far ポインタが使用される。

コンパクト，ラージ，ヒュージメモリモデルでは，**farcalloc** は，**calloc** とまったく同じではありませんがよく似ています。**farcalloc** は unsigned long の引数をとるのに対し，**calloc** は unsigned の引数をとります。
タイニィモデルでは，プログラムを.COM ファイルに変換する場合には，**farcalloc** を使うことはできません。

戻り値　**farcalloc** は，新たに割り当てられたブロックを指すポインタを返します。
要求した大きさのメモリ領域を割り当てることができなかった場合には NULL を返します。

可搬性　**farcalloc** は MS-DOS に特有の関数です。

関連項目　**calloc**，**farcoreleft**，**farfree**，**malloc**

farcoreleft

機能 farヒープの未使用メモリの量を返します。

形式 unsigned long farcoreleft(void);

プロトタイプ alloc.h

解説 **farcoreleft**は，farヒープ中の割り当て済みの最高位ブロックの上にある未使用メモリの量を返します。
タイニィモデルでは，プログラムを.COMファイルに変換する場合には，**farcoreleft**を使うことはできません。

戻り値 **farcoreleft**は，割り当て済みの最高位ブロックとメモリの終わりの間の未使用領域の量を返します。

可搬性 **farcoreleft**はMS-DOSに特有のものです。

関連項目 **coreleft**，**farcalloc**，**farmalloc**

例 **farmalloc**を参照してください。

farfree

機能 far ヒープからブロックを1個解放します。

形式 void farfree(void far ＊ block)；

プロトタイプ alloc.h

解説 **farfree** は，以前に far ヒープに割り当てられたメモリブロックを解放します。

タイニィモデルでは，プログラムを.COM ファイルに変換する場合には，**farfree** を使うことはできません。

スモールおよびミディアムモデルでは，**farmalloc** で割り当てられたブロックは，通常の **free** では解放されません。また **malloc** で割り当てられたブロックは，**farfree** では解放できません。これらのモデルでは，2つのヒープはまったく別物ということになります。

戻り値 戻り値はありません。

可搬性 **farfree** は MS-DOS に特有の関数です。

関連項目 **farcalloc**，**farmalloc**

例 **farmalloc** を参照してください。

farmalloc

機能　far ヒープにメモリを割り当てます。

形式　void far * farmalloc(unsigned long *nbytes*)；

プロトタイプ　alloc.h

解説　**farmalloc** は，far ヒープから *nbytes* バイトの長さのメモリブロックを割り当てます。

far ヒープからの割り当てに関しては，次の点に注意してください。

- ■利用可能な RAM はすべて割り当てることができる。
- ■64K バイトを超えるブロックを割り当てることができる。
- ■割り当てられたブロックをアクセスするには far ポインタが使用される。

コンパクト，ラージ，ヒュージメモリモデルでは，**farmalloc** は，**malloc** とまったく同じではありませんがよく似ています。**farmalloc** は unsigned long の引数をとるのに対し，**malloc** は unsigned の引数をとります。

タイニィモデルでは，プログラムを.COM ファイルに変換する場合には，**farmalloc** を使うことはできません。

戻り値　**farmalloc** は，新たに割り当てられたブロックを指すポインタを返します。要求した大きさのメモリ領域を割り当てることができなかった場合には NULL を返します。

可搬性　**farmalloc** は MS-DOS に特有の関数です。

関連項目　**farcalloc**，**farcoreleft**，**farfree**，**farrealloc**，**malloc**

例

```
/*
   farメモリ管理

   farcoreleft - メモリの残り容量を得る
   farmalloc   - farヒープに領域を割り当てる
   farrealloc  - farヒープのブロックを調整する
   farfree     - farヒープを解放する
*/
#include <stdio.h>
#include <alloc.h>

main()
{
   char far *block;
   long size = 65000;

   /* 残り領域を調べる */
   printf("%lu bytes free¥n", farcoreleft());

   /* その一部を得る */
   block = farmalloc(size);
   if (block == NULL)
   {
      printf("failed to allocate¥n");
      exit(1);
   }
   printf("%lu bytes allocated, ",size);
   printf("%lu bytes free¥n", farcoreleft());

   /* ブロックを調整 */
   size /= 2;
   block = farrealloc(block, size);
   printf("block now reallocated to %lu bytes, ", size);
   printf("%lu bytes free¥n", farcoreleft());

   /* すべてを解放 */
   printf("Free the block¥n");
   farfree(block);
   printf("Block now freed, %lu bytes free¥n", farcoreleft());

}  /* main 終わり */
```

プログラム出力

```
359616 bytes free
65000 bytes allocated, 294608 bytes free
block now reallocated to 32500 bytes, 262100 bytes free
Free the block
Block now freed, 359616 bytes free
```

farrealloc

機能　far ヒープに割り当て済みのブロックを調整します。

形式　void far * farrealloc(void far * *oldblock*, unsigned long *nbytes*) ;

プロトタイプ　alloc.h

解説　**farealloc** は，割り当てられているブロックの大きさを *nbytes* に変更します。必要ならば内容のコピーも行ないます。
far ヒープから割り当てに関しては，次の点に注意してください。

■利用可能な RAM はすべて割り当てることができる。
■64K バイトを超えるブロックを割り当てることができる。
■割り当てられたブロックをアクセスするには far ポインタを使用する。

タイニィモデルでは，プログラムを.COM ファイルに変換する場合には，**farrealloc** を使うことはできません。

戻り値　**farrealloc** は，再割り当てされたブロックのアドレスを返します。これはもとのブロックのアドレスとは異なることもあります。ブロックを再割り当てできなかった場合，**farrealloc** は NULL を返します。

可搬性　**farrealloc** は，MS-DOS に特有の関数です。

関連項目　**farmalloc**，**realloc**

fclose

機能　ストリームをクローズします。

形式　#include <stdio.h>
int fclose(FILE * *stream*);

プロトタイプ　stdio.h

解説　**fclose** は，引数 *stream* に指定されたストリームをクローズします。一般に，ストリームに結びつけられているすべてのバッファの内容は，クローズの前にフラッシュされます。システムが割り当てたバッファはクローズによって解放されます。**setbuf** あるいは **setvbuf** で割り当てられたバッファは自動的には解放されません。

戻り値　**fclose** は，成功した場合0を返します。

可搬性　**fclose** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**close**，**fcloseall**，**fdopen**，**fflush**，**flushall**，**freopen**

例　**fopen** を参照してください。

fcloseall

機能 オープンされているストリームをクローズします。

形式 int fcloseall(void);

プロトタイプ stdio.h

解説 **fcloseall** は，*stdin*，*stdout*，*stdprn*，*stderr*，および *stdaux* を除くすべてのオープンストリームをクローズします。

戻り値 **fcloseall** は，クローズしたストリームの数を返します。エラーがあった場合は EOF を返します。

可搬性 **fcloseall** は UNIX システムで使用できます。

関連項目 **fclose**，**fdopen**，**flushall**，**fopen**，**freopen**

fcvt

機能 浮動小数点数を文字列に変換します。

形式 # include <stdlib>

char * fcvt(double *value*, int *ndig*, int * *dec*, int * *sign*)；

プロトタイプ stdio.h

解説 **fcvt** は，*value* の値を，一番左の有効桁から始まりヌルで終わる *ndig* 桁の文字列に変換し，その文字列を指すポインタを返します。小数点が文字列の先頭から何桁目にあるかは，*dec* を通して間接的に格納されます(*dec* が負の場合は，返された最初の数字の左に小数点があることを意味します)。小数点は文字列そのものには含まれません。結果の符号が負のときは，*sign* が指すワードは0でなくなり，それ以外のときは0となります。
数値は，*ndig* に指定された桁数までに四捨五入されます。

戻り値 **fcvt** の戻り値は静的データを指しているので，**fcvt** が呼ばれるたびに，その内容は上書きされます。

可搬性 **fcvt** は UNIX システムで使用できます。

関連項目 **atof, atoi, atol, ecvt, gcvt**

fdopen

機能　ストリームとファイルハンドルを結びつけます。

形式　# include <stdio.h>

FILE * fdopen(int *handle*, char * *type*)；

プロトタイプ　stdio.h

解説　**fdopen** は，**creat**，**dup**，**dup2**，**open** などによって得られたファイルハンドルとストリームを結びつけます。ストリームのタイプは，オープンされている *handle* のモードと一致しなければなりません。

fdopen で使用される文字列 *type* は，次の値のいずれかです。

r	読み出し専用としてオープンする。
w	書き込み用として作成する。
a	追加用。ファイルの終わりに書き込むようにオープンする。ファイルが存在しない場合は書き込み用として作成する。
r+	既存ファイルを更新（読み出し/書き込み）用として作成する。
w+	新規ファイルを更新用として作成する。
a+	追加用としてオープンする。ファイルの終わりから更新用としてオープンする（ファイルが存在しなければ作成する）。

ファイルがテキストモードでオープンあるいは作成されることを指定するには，*type* の値の最後に *t* をつけます(*rt*，*w*+*t* など)。同様に，バイナリモードを指定するには，*type* の値の最後に *b* をつけます（*wb*，*a*+*b* など）。

t または *b* が *type* に指定されていない場合は，モードはグローバル変数 ***_fmode*** によって決まります。***_fmode*** が O_BINARY にセットされていれば，ファイルはバイナリモードでオープンされます。***_fmode*** が O_TEXT がセットされていれば，テキストモードでオープンされます。定数 O_... は，fcntl.h で定義されています。

ファイルが更新用にオープンされていると，そのストリームに対して入力と出力が行なえなす。しかし，出力のあとで，**fseek** あるいは **rewind** を行なうことなしに入力を行なうことはできません。また入力のあとで，**fseek**, **rewind**, ファイルの終わりを検出する入力のいずれかを行なうことなしに出力を行なうこともできません。

戻り値　成功した場合，**fdopen** は新たにオープンされたストリームを返します。エラーの場合は NULL を返します。

可搬性　**fdopen** は UNIX システムで使用できます。

関連項目　**fclose**，**fopen**，**freopen**，**open**

例

```
#include <stdio.h>
#include <fcntl.h>   /* openが使用するモードの定義が必要 */

main()
{
   int  handle;
   FILE *stream;

   /* ファイルをオープン */
   handle = open("MYFILE.TXT", O_CREAT);

   /* ストリームに結びつける */
   stream = fdopen(handle, "w");
   if (stream == NULL)
      printf("fdopen failed¥n");
   else
   {
      fprintf(stream, "Hello, world¥n");
      fclose(stream);
   }
}
```

feof

機能 ストリーム上でファイルエンドを検出します。

形式

```
#include <stdio.h>
int feof(FILE * stream);
```

プロトタイプ stdio.h

解説 **feof** は，指定した *stream* に対して，ファイル終了標識を調べるマクロです。標識が一度セットされると，**rewind** が呼び出されるか，そのファイルがクローズされるまで，そのファイルに対する読み出し操作はファイル終了標識を返します。

ファイル終了標識は各入力操作でリセットされます。

戻り値 **feof** は，指定のストリームに対する最後の入力操作で，ファイル終了標識が検出された場合に，0でない値を返します。

可搬性 **feof** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **clearerr**，**eof**，**ferror**，**perror**

ferror

機能　　ストリーム上のエラーを検出します。

形式　　#include <stdio.h>
int ferror(FILE * *stream*)；

プロトタイプ　stdio.h

解説　　**ferror** は，指定した *stream* に対して，読み出しおよび書き込みエラーがないかどうかを調べるマクロです。*stream* にエラー標識が一度セットされると，**clearerr** または **rewind** が呼び出されるまで，あるいはそのストリームがクローズされるまで，エラー標識はセットされたままになります。

戻り値　　**ferror** は，指定のストリーム上でエラーを検出すると0でない値を返します。

可搬性　　**ferror** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**clearerr**, **eof**, **feof**, **fopen**, **gets**, **perror**

fflush

機能　ストリームをフラッシュします。

形式　# include <stdio.h>
int fflush(FILE * *stream*)；

プロトタイプ　stdio.h

解説　**fflush** は，指定されたストリームが出力用にオープンされている場合には，*stream* のためのバッファ中の出力を結びつけられているファイルに書き出します。
stream が入力用にオープンされている場合には，バッファの内容がクリアされます。
ストリームは，**fflush** を実行した後もオープンされたままです。**fflush** はバッファリングされないストリームには何の影響も与えません。

戻り値　**fflush** は，成功した場合0を返します。エラーが検出された場合は EOF を返します。

可搬性　**fflush** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**fclose**，**flushall**，**setbuf**，**setvbuf**

fgetc

機能　ストリームから文字を得ます。

形式　# include <stdio.h>
int fgetc(FILE * *stream*)；

プロトタイプ　stdio.h

解説　**fgetc** は，指定されたストリーム上の次の文字を返します。

戻り値　成功した場合，**fgetc** は読み込んだ文字を符号拡張せずに int に変換して返します。ファイルエンドまたはエラーの場合は EOF を返します。

可搬性　**fgetc** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**fgetchar**，**fputc**，**getc**，**getch**，**getchar**，**getche**，**ungetc**，**ungetch**

fgetchar

機能 *stdin* から文字を得ます。

形式 int fgetchar(void)；

プロトタイプ stdio.h

解説 **fgetchar** は，*stdin* から次の1文字を読み込んで返します。
fgetchar は，**fgetc**(*stdin*) と定義されたマクロです。

戻り値 成功した場合，**fgetchar** は読み込んだ文字を符号拡張せずに int に変換して返します。ファイルエンドまたはエラーの場合は EOF を返します。
EOF は **fgetchar** が通常の値として返すものなので，ファイルエンドやエラーを検出するためには，**feof** あるいは **ferror** を使う必要があります。

可搬性 **fgetchar** は UNIX システムで使用できます。

関連項目 **fgetc**，**fputchar**，**getchar**

fgetpos

機能　現在のファイルポインタを得ます。

形式　#include <stdio.h>
int fgetpos(FILE * *stream*, fpos_t * *pos*) ;

プロトタイプ　stdio.h

解説　**fgetpos** は，*stream* に結びつけられているファイルポインタの位置を，*pos* が指している場所に格納します。
fpos_t 型は stdio.h の中で，long 型として typedef 宣言されています。

戻り値　**fgetpos** は成功した場合は0を返し，失敗した場合は0以外の値を返します。

関連項目　**fseek**，**fsetpos**，**ftell**，**tell**

fgets

機能　ストリームから文字列を得る

形式　#include <stdio.h>
char ＊ fgets(char ＊ *s*, int *n*, FILE ＊ *stream*)；

プロトタイプ　stdio.h

解説　**fgets** は，*stream* から文字の並びを読み，文字列 *s* に格納します。読み込みは，(*n*−1)個の文字を読み込むか，改行文字を読み込んだ段階で終わります。**fgets** は，改行文字は文字列に含めません。*s* に読み込まれた最後の文字の後にはヌル文字がつきます。

戻り値　**fgets** は *s* が指している文字列を返します。ファイルエンドあるいはエラーの場合は NULL を返します。

可搬性　**fgets** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**cgets**, **fputs**, **gets**

filelength

機能　ファイルの長さをバイト単位で得ます。

形式

```
#include <io.h>
long filelength(int handle);
```

プロトタイプ　io.h

解説　**filelength** は，*handle* に結びつけられているファイルの長さをバイト単位で返します。

戻り値　成功した場合，**filelength** はファイルの長さを long 型で返します。エラーの場合は-1を返し，***errno*** に次の値をセットします。

EBADF　　ファイル番号が正しくない

関連項目　**fopen，lseek，open**

fileno

機能　　ファイルハンドルを得る

形式　　# include <stdio.h>
int fileno(FILE * *stream*)；

プロトタイプ　　stdio.h

解説　　**fileno** は，引数 *stream* のファイルハンドルを返すマクロです。*stream* が複数のハンドルを持つ場合，**fileno** はそのストリームが最初にオープンされたときに割り当てられたハンドルを返します。

戻り値　　*stream* に結びつけられたファイルハンドルを表わす整数値を返します。

可搬性　　**fileno** は UNIX システムで使用できます。

関連項目　　**fdopen**，**fopen**，**freopen**

findfirst

機能 ディスクディレクトリを探索する。

形式

```
#include <dir.h>
#include <dos.h>
int findfirst(const char *pathname, struct ffblk *ffblk, int attrib);
```

プロトタイプ dir.h

解説 **findfirst** は，DOSのシステムコール0x4Eを使って，ディスクディレクトリの探索を開始します。

pathname は，探索するファイル名の前にオプションのドライブ名，パス名がついたものです。ファイル名の部分には，ワイルドカード（?, ＊）を含めることができます。一致するファイルが見つかった場合は，**ffblk** 構造体にはファイルディレクトリ情報がしまわれます。

ffblk 構造体は次のように定義されています。

```
struct ffblk {
    char ff_reserved[21];      /* MS-DOS で予約 */
    char ff_attrib;            /* アトリビュート */
    int  ff_ftime;             /* ファイルの時刻 */
    int  ff_fdate;             /* ファイルの日付 */
    long ff_fsize;             /* ファイルサイズ */
    char ff_name[13];          /* ファイル名     */
}
```

attrib はMS-DOSのファイル属性バイトであり，探索に際して，選択の基準となるものです。*attrib* はdos.hの中で定義されている次の定数のいずれかとなります。

FA_RDONLY	読み出し専用
FA_HIDDEN	不可視ファイル
FA_SYSTEM	システムファイル
FA_LABEL	ボリュームラベル
FA_DIREC	ディレクトリ
FA_ARCH	アーカイブ

これらの属性についての詳細は，MS-DOSプログラマーズリファレンスマニュアルを参照してください。

findfirst は，DOSのディィスクトランスファアドレス（DTA）に，**ffblk** のアドレスをセットすることに注意してください。

DTAの値を必要とする場合は，**findfirst** の各呼び出しの後で，（**getdta** と **setdta** を使って）DTAをセーブして元に戻す必要があります。

戻り値　*pathname* に一致するファイルが見つかった場合，**findfirst** は0を返します。一致するファイルが見つからない場合や，ファイル名に誤りがある場合は-1を返し，***errno*** に次のいずれかの値をセットします。

ENOENT	パス名やファイル名が見つからない
ENMFILE	もうファイルがない

可搬性　**findfirst** はMS-DOSに特有のものです。

関連項目　**findnext**

例

```
#include <stdio.h>
#include <dir.h>

main()
{
   struct ffblk ffblk;
   int done;
   printf("Directory listing of *.*¥n");
   done = findfirst("*.*", &ffblk, 0);
   while (!done)
   {
      printf("  %s¥n", ffblk.ff_name);
      done = findnext(&ffblk);
   }
}
```

プログラム出力

```
Directory listing of *.*
  FINDFRST.C
  FINDFRST.OBJ
  FINDFRST.MAP
  FINDFRST.EXE
```

findnext

機能　**findfirst** での探索を続けます。

形式　# include <dir.h>
int findnext(struct ffblk * *ffblk*) ;

プロトタイプ　dir.h

解説　**findnext** は，**findfirst** で指定した *pathname* に一致する次のファイルを探すために使用します。*ffblk* は，**findfirst** の呼び出しで情報が格納されたブロックと同じものです。このブロックには，探索を続けるために必要な情報が含まれています。**findnext** は，*pathname* に一致するファイルが対象のディレクトリの中で見つからなくなるまで，1回の呼び出しごとに1つずつファイル名を返します。
findnext は，MS-DOS のディスクトランスファアドレス(DTA)に，**ffblk** のアドレスをセットすることに注意してください。
DTA の値を必要とする場合は，**findnext** の各呼び出しの後で，(**getdta** と **setdta** を使って) DTA をセーブして元に戻す必要があります。

戻り値　*pathname* に一致するファイルが見つかった場合，**findnext** は0を返します。もう一致するファイルが見つからない場合や，ファイル名に誤りがある場合は-1を返し，***errno*** に次のいずれかの値をセットします。

ENOENT	パス名やファイル名が見つからない
ENMFILE	もうファイルがない

可搬性　**findnext** は MS-DOS に特有の関数です。

関連項目　**findfirst**

例　**findfirst** を参照してください。

floor

機能　小数点以下の切り捨てを行ないます。

形式　#include <math.h>
double floor(double *x*);

プロトタイプ　math.h

解説　**floor** は，*x* を越えない最大の整数を返します。

戻り値　**floor** は，見つかった整数を **double** で返します。

可搬性　**floor** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**ceil**，**fmod**

flushall

機能　すべてのバッファをフラッシュします。

形式　int flushall(void)；

プロトタイト　stdio.h

解説　**flushall** は，オープンされている入力ストリームに結合しているすべてのバッファのすべてをクリアし，出力ストリームに結びつけられているすべてのバッファの内容を対応するファイルに書き出します。**flushall** の後で読み込みを行なうと，入力ファイルからバッファにデータを新たに読み込むことになります。
flushall の実行後も，ストリームはオープンされたままです。

戻り値　**flushall** は，オープンされている入力および出力ストリームの数を返します。エラーがあった場合は EOF を返します。

可搬性　**flushall** は UNIX システムで使用できます。

関連項目　**fclose**，**fcloseall**，**fflush**

fmod

機能　x を y で割った剰余を計算します。

形式　

```
# include <math.h>
double fmod(double x, double y);
```

プロトタイプ　math.h

解説　**fmod** は，x を y で割った余り（$x=iy+f$ および$0=<f<y$ を満たす f。i は整数）を計算します。

戻り値　**fmod** は，剰余 f （$x = iy + f$）を返します。

可搬性　**fmod** は ANSI C と互換性があります。

関連項目　**ciel**，**floor**，**modf**

fnmerge

機能 ファイル名を作ります。

形式 #include <dir.h>

void fnmerge(char * *path*, const char * drive, const char * dir,
const char * *name*, const char * *ext*);

プロトタイプ dir.h

解説 **fnmerge** は，要素を組み合わせてパス名を作り出します。新しいパス名は次の形式になります。

X：¥DIR¥SUBDIR¥NAME.EXT

このファイル名は次のようにして構成されます。

■ X は，*drive* によって与えられる
■ ¥DIR¥SUBDIR は，*dir* によって与えられる
■ NAME.EXT は，*name* と *ext* によって与えられる

fnmerge は，*path* には構成されたパス名を格納するのに充分な領域があるものとみなします。構成されるパス名の最大長は MAXPATH です。MAXPATH は dir.h で定義されています。
fnmerge と **fnsplit** は，ちょうど逆の操作を行ないます。つまり，あるパス名を **fnsplit** で分解して，**fnmerge** で各要素からパス名を組み立てると，元のパスが復元されます。

戻り値 ありません。

可搬性 **fnmerge** は MS-DOS でのみ使用できます。

関連項目 **fnsplit**

例

```
#include <stdio.h>
#include <dir.h>

char drive[MAXDRIVE];
char dir[MAXDIR];
char file[MAVFILE];
char ext[MAXEXT];

main()
{
   char s[MAXPATH], t[MAXPATH];
   int flag;

   for (;;)
   {
      printf("> ");            /* さらに入力があるかぎり */
      if (!gets(s)) break;     /* 入力のプロンプトを表示 */
      if (!gets[0]) break;
      flag = fnsplit(s,drive,dir,file,ext);

      /* 要素を表示 */
      printf(" drive: %s, dir: %s, file: %s, ext: %s, ",
               drive, dir, file, ext);
      printf("flags: ");
      if (flag & DRIVE)
         printf(":");
      if (flag & DIRECTORY)
         printf("d");
      if (flag & FILENAME)
         printf("f");
      if (flag & EXTENTION)
         printf("e");
      printf("¥n");

      /* 各要素を再構成して元の文字列と比較 */
      fnmerge(t,drive,dir,file,ext);
      if (strcmp(t.s) != 0)       /* 起こるはずはない */
         printf(" --> strings are different!");
   }
}
```

プログラム出力

```
> C:¥TURBOC¥FN.C
  drive: C:, dir: ¥TURBOC¥, file: FN, ext: .C,
  flags: :dfe
> FILE.C
  drive: , dir: , file: FILE, ext: .C, flags: fe
> ¥TURBOC¥SUBDIR|NOEXT.
  drive: , dir: ¥TURBOC¥SUBDIR¥, file: NOEXT,
  ext: ., flags: dfe
> C:MYFILE
  drive: C:, dir: , file:MYFILE, ext: , flags :f
> ^Z
```

fnsplit

機能　フルパス名を要素に分割します。

形式　#include <dir.h>

int fnsplit(const char * *path*, char * *drive*, char * *dir*,
　　　　　char * *name*, char * *ext*);

プロトタイプ　dir.h

解説　**fnsplit** は，次の形式の文字列が表わすファイルのフルパス名（*path*）を，4つの要素に分割します。

X：¥DIR¥SUBDIR¥NAME.EXT

各要素はそれぞれ *drive*，*dir*，*name*，*ext* が指す文字列に格納されます（5つの引数すべてを渡さなければなりません。要素には NULL を指定することができますが，その場合，その要素の解析が行なわれるだけでどこにも格納されません）。

これら文字列の最大長は，定数 MAXDRIVE，MAXDIR，MAXPATH，MAXNAM（dir.h の中で定義されています）で与えられます。これらの中にはヌル文字のスペースも含まれます。

定数	（最大長）	文字列
MAXPATH	(80)	*path*
MAXDRIVE	(3)	*drive*：コロン（：）も含む
MAXDIR	(66)	*dir*：先頭と最後の円記号（¥）も含む
MAXFILE	(9)	*name*
MAXNET	(5)	*ext*：先頭のピリオド（.）も含む

fnsplit は，NULL でない要素を格納するための充分な大きさのスペースがあることを仮定しています。

fnsplit が *path* を分割するときには，次の規則が適用されます。

- *drive* には，コロンが含まれる（C：，A：など）
- *dir* には，先頭と最後の円記号も含まれる（¥turbo¥include, ¥source¥など）
- *name* には，ファイル名が含まれる
- *ext* には，拡張子の前のピリオドが含まれる

fnsplit と **fnmerge** はちょうど逆の操作を行ないます。ある *path* を **fnsplit** で分割して，分割した各要素を **fnmerge** で組み立てると，元の *path* ができあがります。

戻り値　**fnsplit** は，フルパス名のどの要素が *path* の中に含まれているかを表わす（dir.h で定義されている5個のフラグからなる）整数を返します。各フラグと対応する要素は次のようになっています。

EXTENSION	拡張子
FILENAME	ファイル名
DIRECTORY	ディレクトリ（およびサブディレクトリ）
DRIVE	ドライブ指定（dir.h を参照）
WILDCARD	ワイルドカード（＊または?）

可搬性　**fnsplit** は MS-DOS システムでのみ使用可能です。

関連項目　**fnmerge**

例　**fnmerge** を参照してください。

fopen

機能　ストリームをオープンする。

形式　#include <stdio.h>

FILE * fopen(const char * *filename*, const char * *mode*);

プロトタイプ　stdio.h

解説　**fopen** は *filename* で指定されたファイルをオープンし，ストリームと結びつけます。**fopen** は以後の操作でそのストリームを識別するのに使用されるポインタを返します。

fopen で使用される文字列 *mode* は，次の値のいずれかです。

r	読み出し専用としてオープンする。
w	書き込み用として作成する。
a	追加用。ファイルの終わりに書き込むようにオープンする。ファイルが存在しない場合は書き込み用として作成する。
r+	既存ファイルを更新（読み出し/書き込み）用として作成する。
w+	新規ファイルを更新用として作成する。
a+	追加用としてオープンする。ファイルの終わりから更新用としてオープンする（ファイルが存在しなければ作成する）。

ファイルがテキストモードでオープンあるいは作成されることを指定するには，*mode* の値の最後に *t* をつけます(*rt*, *w*+*t* など)。同様に，バイナリモードを指定するには，*mode* の値の最後に *b* をつけます(*wb*, *a*+*b* など)。**fopen** では，文字 *t* または *b* を，*mode* 文字列中の文字と＋の間に入れてもかまいません。たとえば，*rt*+は *r*+*t* と同じ意味をもちます。

t または *b* が *mode* に指定されていない場合は，モードはグローバル変数 ***_fmode*** によって決まります。***_fmode*** が O_BINARY にセットされていれば，ファイルはバイナリモードでオープンされます。***_fmode*** に O_TEXT がセットされていれば，テキストモードでオープンされます。定

数 O_... は，fcntl.h で定義されます。

ファイルが更新用にオープンされていると，そのストリームに対して入力と出力が行なえなす。しかし，出力のあとで，**fseek** あるいは **rewind** を行なうことなしに入力を行なうことはできません。また入力のあとで，**fseek, rewind,** ファイルの終わりを検出する入力のいずれかを行なうことなしに出力を行なうこともできません。

戻り値　成功した場合，**fopen** は新たにオープンされたストリームへのポインタを返します。エラーの場合は NULL を返します。

可搬性　**fopen** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**creat, dup, fclose, fdopen, ferror,** ***_fmode***（変数），**fopen, fread, freopen, fseek, fwrite, open, rewind, setbuf, setmode**

例

```
/* AUTOEXEC.BAT ファイルのバックアップを作成するプログラム */

#include <stdio.h>

main()
{
   FILE *in, *out;

   if ((in = fopen("¥¥AUTOEXEC.BAT", "rt")) == NULL)
   {
      fprint(stderr, "Cannot open input file.¥n");
      return (1);
   }

   if ((out = fopen("¥¥AUTOEXEC.BAK","wt")) == NULL)
   {
      fprint(stderr, "Cannot open output file.¥n");
      return (1);
   }

   while (!feof(in))
      fputc(fgetc(in), out);

   fclose(in);
   fclose(out);
}
```

FP_OFF

機能 far アドレスのオフセット値を得ます。

形式 # include <dos.h>
unsigned FP_OFF(*farpointer*);

プロトタイプ dos.h

解説 **FP_OFF** マクロは，far ポインタ *farpointer* のオフセット値を得るために使用されます。

戻り値 **FP_OFF** は，オフセット値を表わす符号なし整数値を返します。

関連項目 **FP_SEG**，**MK_FP**，**movedata**，**segread**

例

```
#include <stdio.h>
#include <dos.h>

main ()
{
   char far *ptr;
   unsigned seg, off;

   ptr = MK FP(0xB000,0);
   seg = FP_SEG(ptr);
   off = FP_OFF(ptr);
   printf("far ptr %Fp, segment %04x offset %04x¥n",ptr,seg,off);
}
```

プログラム出力

```
far ptr B000:0000, segment b000, offset 0000
```

_fpreset

機能　浮動小数点数学パッケージを再初期化します。

形式　void _fpreset(void);

プロトタイプ　float.h

解説　**_fpreset** は，浮動小数点数学パッケージの再初期化を行ないます。この関数は通常，**system**，**exec...**，**spawn...**関数とともに使用されます。

注意：MS-DOS のバージョン2.*x* や3.*x* より前のバージョンの下で，プログラム中で8087/80287コプロセッサが使用されている場合，(**system**，**exec...**，**spawn...**関数で実行した)子プロセスは，親プロセスの浮動小数点状態を変更する場合があります。

8087/80287を使う場合は次のような点に注意してください。

- ■浮動小数点の式を評価している間は，**system**，**exec...**，**spawn...**関数は呼び出さない。
- ■子プロセスが8087/80287で浮動小数点操作を行なった可能性がある場合は，**system**，**exec...**，**spawn...**関数の後で **_fpreset** を呼んで，浮動小数点状態をリセットすべきである。

戻り値　戻り値はありません。

関連項目　**_clear87**，**_control87**，**exec...**，**spawn...**，**_status87**，**system**

fprintf

機能　ストリームに書式つき出力を行ないます。

形式　# include <stdio.h>

int fprintf(FILE * *stream*, const char * *format* [, argument,...]);

プロトタイプ　stdio.h

解説　**fprintf** は，*format* によって指される書式文字列中の書式指定を，*format* の後に続く各引数に適用し，書式化されたデータを指定されたストリームに出力します。書式指定の数は，後に続く引数と同じだけなければなりません。

書式指定に関する詳細な情報については，**printf** の解説を参照してください。

戻り値　**fprintf** は，出力したバイト数を返します。エラーの場合には EOF を返します。

可搬性　**fprintf** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**cprintf**, **fscanf**, **printf**, **putc**, **sprintf**

例　**printf** を参照してください。

FP_SEG

機能　far アドレスのセグメント値を得ます。

形式　# include <dos.h>

unsigned FP_SEG(*farpointer*);

プロトタイプ　dos.h

解説　**FP_SEG** マクロは，far ポインタ *farpointer* のセグメント値を得るために使用されます。

戻り値　**FP_SEG** は，セグメント値を表わす符号なし整数値を返します。

関連項目　**FP_OFF**，**MK_FP**

例　**FP_OFF** を参照してください。

fputc

機能　ストリームへ1文字を出力します。

形式　#include <stdio.h>
int fputc(int c, FILE * *stream*);

プロトタイプ　stdio.h

解説　**fputc** は，文字 *c* を指定したストリーム *stream* へ出力します。

戻り値　成功した場合，**fputc** は文字 *c* を返します。エラーの場合は EOF を返します。

可搬性　**fputc** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**fgetc**, **putc**

fputchar

機能　*stdout* へ1文字出力します。

形式　# include <stdio.h>
int fputchar(int *c)*；

プロトタイプ　stdio.h

解説　**fputchar** は，文字 *c* を *stdout* に出力します。
fputchar(*c*)は，**fputc**(*c*,*stdout*)と同じです。

戻り値　成功した場合，**fputchar** は文字 *c* を返します。エラーの場合は EOF を返します。

可搬性　**fputchar** は UNIX システムで使用できます。

関連項目　**fgetchar**，**putchar**

fputs

機能 ストリームに文字列を出力します。

形式 # include <stdio.h>

int fputs(const char * *s*, FILE * *stream*);

プロトタイプ stdio.h

解説 **fputs** は，ヌル文字で終わる文字列 *s* を，指定の出力ストリーム *stream* へコピーします。改行文字はつけず，また最後のヌル文字はコピーされません。

戻り値 **fputs** は，成功した場合は最後に書いた文字を返し，エラーの場合は EOF を返します。

可搬性 **fputs** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目 **fgets**, **gets**, **puts**

fread

機能　ストリームからデータを読み込みます。

形式　# include <stdio.h>

size_t fread(void * *ptr*, size_t *size*, size_t *n*, FILE * *stream*)；

プロトタイプ　stdio.h

解説　**fread** は，*n* 個のデータ（各データは *size* バイトの大きさ）を指定の入力ストリームから読み込んで，*ptr* が指すブロックに格納します。

読み込まれるバイト数の合計は（*n*×*size*）となります。

ptr は，どんなオブジェクトでも指すことができるように宣言されています。

戻り値　成功した場合，**fread** は実際に読み込んだデータの個数を返します（バイト数ではありません）。ファイルエンドまたはエラーの場合には，データ数より小さな値（0のこともあります）を返します。

可搬性　**fread** はすべて UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**fopen**，**fwrite**，**printf**，**read**

free

機能　割り当て済みのブロックを解放します。

形式　void free(void ＊ *block*)；

プロトタイプ　stdlib.h，alloc.h

解説　**free** は，**calloc**, **malloc**, **realloc** の呼び出しで以前に確保されたブロックを解放します。*block* には，そのメモリブロックの先頭バイトのアドレスが入っていなければなりません。

戻り値　戻り値はありません。

可搬性　**free** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**calloc**，**freemem**，**malloc**，**realloc**，**strdup**

freemem

機能　割り当て済みのDOSメモリブロックを解放します。

形式　int freemem(unsigned *segx)*;

プロトタイプ　dos.h

解説　**freemem**は，以前の**allocmem**の呼び出しで確保されたメモリブロックを解放します。*segx*には，そのブロックのセグメントアドレスを指定します。

戻り値　**freemem**は，成功した場合には0を返します。エラーの場合には-1を返し，***errno***にENOMEM（メモリが不足した）をセットします。

関連項目　**allocmem**，**free**

freopen

機能　ストリームを置き換えます。

形式　# include <stdio.h>

FILE * freopen(const char * *filename*, const char * *mode*,
　　　　FILE * *stream*);

プロトタイプ　stdio.h

解説　**freopen** は，オープンストリーム *stream* を，指定のファイルに置きかえます。*stream* は，そのオープンが成功したかどうかに関係なくクローズされます。

freopen は，*stdin*，*stdout*，*stderr* に結びつけられているファイルを変更するのに便利です。

freopen の呼び出しで使用される文字列 *mode* は，次の値のいずれかです。

r	読み出し専用としてオープンする。
w	書き込み用として作成する。
a	追加用。ファイルの終わりに書き込むようにオープンする。ファイルが存在しない場合は書き込み用として作成する。
r+	既存ファイルを更新（読み出し/書き込み）用として作成する。
w+	新規ファイルを更新用として作成する。
a+	追加用としてオープンする。ファイルの終わりから更新用としてオープンする（ファイルが存在しなければ作成する）。

指定ファイルがテキストモードでオープンあるいは作成されることを指定するには，*mode* の値の最後に *t* をつけます（*rt*，*w*+*t* など）。同様に，バイナリモードを指定するには，*mode* の値の最後に *b* をつけます（*wb*，*a*+*b* など）。

t または *b* が *mode* に指定されていない場合は，モードはグローバル変数 ***_fmode*** によって決まります。***_fmode*** が O_BINARY にセットされてい

れば，ファイルはバイナリモードでオープンされます。_*fmode* がO TEXT がセットされていれば，テキストモードでオープンされます。定数O_...は，fcntl.h で定義されます。
ファイルが更新用にオープンされていると，そのストリームに対して入力と出力が行なえなす。しかし，出力のあとで，**fseek** あるいは **rewind** を行なうことなしに入力を行なうことはできません。また入力のあとで，**fseek, rewind,** ファイルの終わりを検出する入力のいずれかを行なうことなしに出力を行なうこともできません。

戻り値　成功した場合，**freopen** は引数 *stream* を返します。エラーの場合は NULL を返します。

可搬性　**freopen** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**fclose**，**fdopen**，**fopen**，**open**，**setmode**

例　**fopen** を参照してください。

frexp

機能　double の変数を仮数部と指数部に分割します。

形式　#include <math.h>

double frexp(double *x*, int * *exponent*)；

プロトタイプ　math.h

解説　**frexp** は，引数 x（もとの double 値）から，$x = m \times 2^n$ となるような仮数 m（0.5以上1未満の double）と整数値 n を計算します。**frexp** は，*exponent* が指す整数に n の値を格納します。

戻り値　**frexp** は仮数 m を返します。

frexp のエラー処理は，**matherr** 関数を使って変更することができます。

可搬性　**frexp** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**exp**，**ldexp**

fscanf

機能 ストリームから書式つき入力を行ないます。

形式 # include <stdio.h>

int fscanf(FILE * *stream*, const char * *format* [, *address*,...]);

プロトタイプ stdio.h

解説 **fscanf** は，一連の入力フィールドをスキャンして，一度に1文字ずつストリームから文字を読み込みます。次に，引数 *format* によって指される書式文字列中の書式指定にしたがって，各フィールドを書式化します。最後に，*format* の後に続く各引数が示しているアドレスに，書式化した入力を格納していきます。書式文字列中の書式指定の個数は，その後に続くアドレスの数と同じでなければなりません。

書式指定も含めて，詳細な情報については **scanf** の解説を参照してください。

fscanf は，通常のフィールドの終了（ホワイトスペース）に達する前に，フィールドのスキャンをやめる場合があります。また，いくつかの理由から入力を終了してしまうこともあります。こうした問題については **scanf** の解説を参照してください。

戻り値 **fscanf** は，正しくスキャンし，変換し，格納した入力フィールドの数を返します。戻り値には，値を格納しなかった入力フィールドの数は含まれません。

ファイルエンドを読み込もうとした場合は，戻り値は EOF になります。値を格納したフィールドがなかった場合は，戻り値は0になります。

可搬性 **fscanf** は UNIX システムで使用でき，K&R で定義されています。ANSI C とも互換性があります。

関連項目 **atof**, **cscanf**, **fprintf**, **printf**, **scanf**, **sscanf**, **vfscanf**, **vscanf**, **vsscanf**

fseek

機能　ストリーム上のファイルポインタを移動します。

形式　# include <stdio.h>
int fseek(FILE * *stream*, long int *offset*, int *whence*);

プロトタイプ　stdio.h

解説　**fseek**は，*stream*に結びつけられているファイルポインタを，*whence*で指定されたファイルの位置から，*offset*バイト離れたところに位置づけます。テキストモードのストリームでは，*offset*は0か，**ftell**によって返された値でなければなりません。
*whence*は，0，1，2のいずれかでなければなりません。この3つの定数は次のようなシンボリック定数で表されます（stdio.hの中で定義されています）。

whence	**位置**
SEEK_SET (0)	ファイルの初め
SEEK_CUR (1)	現在のファイルポインタの位置
SEEK_END (2)	ファイルの終わり

fseekは，**ungetc**がプッシュバックした文字は捨て去ります。
fseekは，ストリームI/Oに対して使用します。ファイルハンドルI/Oには**lseek**を使ってください。
更新用にオープンされたファイルに対しては，**fseek**を呼び出した後は，入力・出力どちらでも行なうことができます。

戻り値　成功した場合，**fseek**は0を返し，エラーの場合は0でない値を返します。

可搬性　　**fseek** はすべての UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**fgetpos**，**fopen**，**fsetpos**，**ftell**，**lseek**，**rewind**，**setbuf**，**tell**

例

```
#include <stdio.h>

/* ファイルストリーム内のバイト数を返す */
long filesize(FILE *stream)
{
   long curpos,length;

   curpos = ftell(stream);
   fseek(stream, OL, SEEK_END);
   length = ftell(stream);
   fseek(stream, curpos, SEEK_SET);
   return(lenght);
}

main()
{
   FILE *stream;

   stream = fopen("MYFILE.TXT","r");
   printf("filesize of MYFILE.TXT is %ld"
          "bytes¥n",filesize(stream));
}
```

プログラム出力

```
filesize of MYFILE.TXT is 15 bytes
```

fsetpos

機能　ストリーム上のファイルポインタの位置を設定します。

形式

```
#include <stdio.h>
int fsetpos(FILE * stream, const fpos_t * pos);
```

プロトタイプ　stdio.h

解説　**fsetpos**は，*stream*に結びつけられているファイルポインタを新しい位置にセットします。新しい位置は，その*stream*に対して**fgetpos**を前回呼び出したときに得られた値です。**fsetpos**は，*stream*が指しているファイルのファイル終了標識をクリアし，そのファイルに対する**ungetc**の効果を無効にします。**fgetpos**を呼び出した後は，そのファイルに対して，入力または出力を行なうことができます。

戻り値　**fsetpos**は成功した場合0を返します。失敗した場合は0以外の値を返し，***errno***に0以外の値をセットします。

可搬性　**fsetpos**はANSI Cと互換性があります。

関連項目　**fgetpos**，**fseek**，**ftell**

fstat

機能　オープンファイルの情報を得ます。

形式　# include <sys¥stat.h>
int fstat(int *handle*, struct stat * *statbuf*);

プロトタイプ　sys¥stat.h

解説　**fstat** は，*handle* に結びつけられているオープンファイルまたはディレクトリにに関する情報を，**stat** 構造体の中に格納します。
statbuf は，**stat** 構造体(sys¥stat.h の中で定義されている)を指しています。その構造体の中には次のようなフィールドが含まれています。

st_mode	オープンファイルのモードに関する情報を与えるビットマスク
st_dev	そのファイルがあるディスクのドライブ番号，またはファイルがデバイス上にある場合にはファイルハンドル
st_rdev	*st_dev* と同じ
st_nlink	整定数1にセット
st_size	オープンファイルの大きさ（バイト数）
st_atime	オープンファイルの一番最近修正された日時
st_mtime	*st_atime* と同じ
st_ctime	*st_atime* と同じ

stat 構造体には，これ以外にフィールドが3つありますが，MS-DOS では意味を持たないので省略します。
ビットマスクはオープンファイルのモードに関する情報を含んでおり，各ビットは次のような意味を持っています。
次のビットのうち，いずれか1つがセットされます。

S_IFCHR	*handle* がデバイスを参照している場合
S_IFREG	通常のファイルが *handle* によって参照されている場合

次の一方または両方のビットがセットされます。

S_IWRITE	ユーザが書き込み許可を得ている場合
S_IREAD	ユーザが読み込み許可を得ている場合

ビットマスクには，さらに読み込み/書き込みビットも含まれています。これらはファイルの許可モードにしたがってセットされます。

戻り値　**fstat** は，そのオープンファイルに関する情報をうまく得られた場合は0を返します。エラーの場合（情報が得られなかった）は-1を返し，***errno*** にEBADF（ファイルハンドルが正しくない）をセットします。

関連項目　**access**，**chmod**，**stat**

ftell

機能 現在のファイルポインタを返します。

形式 # include <sdtio.h>

long int ftell(FILE * *stream*);

プロトタイプ stdio.h

解説 **ftell** は，*stream* 中の現在のファイルポインタの位置を返します。オフセットはファイルの先頭からバイト単位で数えます。

ftell が返した値は，それに続く **fseek** の呼び出しで使用することができます。

戻り値 **ftell** は，成功した場合，現在のファイルポインタ位置を返します。エラーの場合は-1L を返し，***errno*** に正の値をセットします。

可搬性 **ftell** はすべての UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **fgetpos**，**fseek**，**fsetpos**，**lseek**，**rewind**，**tell**

例 **fseek** を参照してください。

ftime

機能　現在の時刻を **timeb** 構造体に格納します。

形式　# include <sys¥timeb.h>
void ftime(struct timeb * *buf*)；

プロトタイプ　sys¥timeb.h

解説　**ftime** は現在の時刻を決定し，*buf* によって指される **timeb** 構造体の各フィールドを埋めます。**timeb** 構造体は，*time*，*millitm*，*timezone*，*dstflag* の4つのフィールドからなっています。

- *time* フィールドは，グリニッジ標準時(GMT)1970年1月1日00：00：00からの経過秒数を与えます。
- *millitm* フィールドは，ミリ秒単位で *time* の端数を表わします。
- *timezone* フィールドは，GMT とローカルタイムとの差を分単位で示します。この値は GMT から西に向かって計算されます。**ftime** はこの値を，**tzset** 関数によってセットされたグローバル変数 ***timezone*** から取り出します。
- *dstflag* フィールドは，その時間帯において夏時間が有効でない場合にはゼロに，その時間帯において夏時間が有効な場合にはゼロでない値にセットされます。

注意：**ftime** は **tzset** を呼び出します。**ftime** を使用することが明らかな場合には，**tzset** を呼び出す必要はありません。

戻り値　ありません。

互換性　**ftime** は UNIX システム V で使用できます。

関連項目　**asctime**，**ctime**，**gmtime**，**localtime**，**stime**，**time**，**tzset**

例

```
#include <stdio.h>
#include <sys¥timeb.h>

main()
{
    struct timeb buf;

    ftime(&buf);
    printf("%ld Seconds since 1-1-70 GMT¥n", buf.time);
    printf("plus %d milliseconds¥n", buf.millitm);
    printf("%d Minutes from GMT¥n", buf.timezone);
    printf("Daylight savings %s in effect¥n",
           buf.dstflag ? "is" : "is not");
}
```

fwrite

機能　ストリームにデータを書きます。

形式　# include <stdio.h>
size_t fwrite(const void * *ptr*, size_t *size*, size_t *n*,FILE * *stream*);

プロトタイプ　stdio.h

解説　**fwrite** は，*n* 個のデータ（各データは *size* バイトの大きさ）を指定の出力ストリームに追加します。*ptr* が指すデータが出力されます。
書き込まれるバイト数の合計は（*n*×*size*）となります。
ptr は，どんなオブジェクトでも指せるように宣言されています。

戻り値　成功した場合，実際に書き込んだデータの個数を返します（バイト数ではありません）。エラーがあった場合には，データ数より小さな値を返します。

可搬性　**fwrite** はすべて UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**fopen**，**fread**

gcvt

機能　浮動小数点数を文字列に変換します。

形式　# include <dos.h>

char * gcvt(double *value*, int *ndec*, char * *buf*) ;

プロトタイプ　stdio.h

解説　**gcvt** は，*value* をヌルで終わる ASCII 文字列に変換し，その文字列を *buf* に格納します。**gcvt** は，可能であれば，FORTRAN の F 変換に則して，*ndec* 桁の有効数字を生成します。F 変換が不可能な場合は，**printf** の E 形式で（プリント可能なように）生成します。後ろに続く意味のない0は文字列に含まれません。

戻り値　**gcvt** は，*buf* が指す文字列のアドレスを返します。

可搬性　**gcvt** は UNIX システムで使用できます。

関連項目　**ecvt**, **fcvt**

geninterrupt

機能　ソフトウェア割り込みを生成します。

形式
```
#include <dos.h>
void geninterrupt(int intr_num);
```

プロトタイプ　dos.h

解説　**geninterrupt** マクロは，*intr_num* に指定された割り込みに対するソフトウェアトラップを引き起こす働きをします。呼び出し後のすべてのレジスタの状態は，呼び出された割り込みに依存します。

戻り値　ありません。

可搬性　**geninterrupt** は80x86アーキテクチャに特有のものです。

関連項目　**bdos, bdosptr, getvect, int86, int86x, intdos, intdosx, intr**

getc

機能　ストリームから1文字を得ます。

形式　# include <stdio.h>
int getc(FILE ＊ *stream*)；

プロトタイプ　stdio.h

解説　**getc** は，指定の入力ストリームから次の1文字を読み込んで，そのストリームのファイルポインタをインクリメントするマクロです。

戻り値　成功した場合，**getc** は読み込んだ文字を符号拡張せずに int 型に変換して返します。ファイルエンドまたはエラーの場合は EOF を返します。

可搬性　**getc** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**fgetc**, **getch**, **getchar**, **getche**, **gets**, **putc**, **putchar**, **ungetc**

getcbrk

機能　コントロールブレークの設定を得ます。

形式　int getcbrk(void);

プロトタイプ　dos.h

解説　**getcbrk** は，DOS のシステムコール0x33を使って，現在のコントロールブレークチェックの設定を返します。

戻り値　**getcbrk** は，コントロールブレークチェックがオフなら0，オンなら1を返します。

可搬性　**getcbrk** は MS-DOS に特有の関数です。

関連項目　**ctrbrk**, **setcbrk**

getch

機能　キーボードからエコーバックなしで文字を読み込みます。

形式　int getch(void)；

プロトタイプ　conio.h

解説　**getch** は，コンソールから直接1文字を読み込む関数です。エコーバックはしません。**getch** は *stdin* を使用します。

戻り値　**getch** は，キーボードから読み込んだ文字を返します。エラーの戻り値はありません。

可搬性　**getch** は MS-DOS に特有の関数です。

関連項目　**cgets, fgetc, getc, getchar, getche, getpass, kbhit, putch, ungetch**

getchar

機能　*stdin* から文字を読み込みます。

形式　# include <stdio.h>
int getchar(void);

プロトタイプ　stdio.h

解説　**getchar** は，ストリーム *stdin* 上の次の1文字を返すマクロです。これは，**getc**(*stdin*) と定義されています。

戻り値　成功した場合，**getchar** は読み込んだ文字を符号拡張せずに int 型に変換して返します。ファイルエンドまたはエラーの場合は EOF を返します。

可搬性　**getchar** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**fgetc**, **fgetchar**, **getc**, **getch**, **getche**, **putc**, **putchar**, **ungetc**

getche

機能　キーボードからエコーつきで文字を読み込みます。

形式　int getche(void);

プロトタイプ　conio.h

解説　**getche** は，キーボードから1文字を読み込み，現在のテキストウィンドウにエコーバックします。**getche** は BIOS を使用します。

戻り値　**getche** は読み込んだ文字を返します。

可搬性　**getche** は MS-DOS に特有の関数です。

関連項目　**cgets, cscanf, fgetc, getc, getch, getchar, kbhit, putch, ungetch**

getcurdir

機能　指定したドライブのカレントディレクトリを得ます。

形式　int getcurdir(int *drive*, char * *directory*);

プロトタイプ　dir.h

解説　**getcurdir** は，指定ドライブのカレント作業ディレクトリの名前を得ます。
drive にはドライブ番号（0=カレント，1=A,...）を指定します。
directory は，ヌルで終わるディレクトリ名が格納される，長さ MAXDIR のメモリ領域を指します。ディレクトリ名にはドライブ名は含まれず，先頭に円記号¥もつきません。

戻り値　**getcurdir** は，成功した場合は0を返し，エラーの場合は-1を返します。

可搬性　**getcurdir** は MS-DOS に特有の関数です。

関連項目　**chdir**, **getcwd**, **getdisk**, **mkdir**, **rmdir**

例

```
#include <dir.h>
#include <stdio.h>
#include <string.h>

char *current_directory(char *path)
{
   strcpy(path, "X:¥¥");
   path[0] = 'A' + getdisk();
   getcurdir(0, pash+3);
   return(path);
}

main()
{
   char curdir[MAXPATH];

   current_directory(curdir);
   printf("The current directory is %s¥n, curdir);
}
```

プログラム出力

```
The current directory is A:¥TURBOC
```

getcwd

機能　現在の作業ディレクトリを得ます。

形式　char * getcwd(char * *buf*, int *buflen*) ;

プロトタイプ　dir.h

解説　**getcwd** は，現在の作業ディレクトリ（ドライブ名も含む）のフルパス名を得る関数です。最大 *buflen* 文字まで *buf* の中に格納します。フルパス名の長さ（最後のヌル文字まで）が *buflen* より長い場合はエラーになります。*buf* が NULL の場合は自動的に **malloc** が呼び出され，*buflen* バイト分のバッファが割り当てられます。後で，**getcwd** が返した値を引数として **free** 関数を呼び出せば，このバッファを解放することができます。

戻り値　**getcwd** は次のような値を返します。

- *buf* が NULL でなかった場合，成功すれば *buf* を返し，失敗すれば NULL を返します。
- *buf* が NULL だった場合は，割り当てたバッファへのポインタを返します。

エラーの場合は NULL を返し，グローバル変数 ***errno*** に次のいずれかの値をセットします。

ENODEV	指定のデバイスがない
ENOMEM	メモリが不足した
ERANGE	結果が範囲を越えた

可搬性　***getcwd*** は MS-DOS に特有の関数です。

関連項目　**chdir**, **getcurdir**, **getdisk**, **mkdir**, **rmdir**

getdate

機能　MS-DOS の日付を得ます。

形式　# include <dos.h>

void getdate(struct date * *datep*);

プロトタイプ　dos.h

解説　**getdate** は，システムの現在の日付を，*datep* が指す **date** 構造体に格納します。

date 構造体は次のように定義されています。

```
struct date {
   int da_year;         /* 年 */
   char da_day;         /* 日 */
   char da_mon;         /* 月 */
}
```

戻り値　ありません。

可搬性　**getdate** は MS-DOS に特有の関数です。

関連項目　**ctime**, **gettime**, **setdate**, **settime**

例

```
#include <stdio.h>
#include <dos.h>

main()
{
   struct date today;
   struct time now;

   getdate(&today);
   printf("Today's date is %d/%d/%d¥n",
           today.da_year, today.da_mon, today.da_day);

   gettime(&now);
   printf("The time is %02d:%02d:%02d.%02d¥n",
           now.ti_hour, now.ti_min, now.ti_sec, now.ti_hund);
}
```

プログラム出力

```
Today's date is 1/1/1988
The time is 17:08:22.00
```

getdfree

機能　ディスクの未使用領域の大きさを得ます。

形式　# include <dos.h>

void getdfree(unsigned char *drive*, struct dfree ＊ *dtable*)；

プロトタイプ　dos.h

解説　**getdfree** は，*drive* で指定したドライブ（0＝デフォルト，1＝A,...）のディスクに関する情報を，*dtable* が指す **dfree** 構造体に格納します。

dfree 構造体は次のように定義されています。

```
struct dfree {
   unsigned df_avail;       /* 使用可能なクラスタ数 */
   unsigned df_total;       /* 総クラスタ数 */
   unsigned df_bsec;        /* 1セクタのバイト数 */
   unsigned df_sclus;       /* 1クラスタのセクタ数 */
};
```

戻り値　**getdfree** は値を返しません。エラーの場合は，**dfree** 構造体の *df_sclus* に -1がセットされます。

可搬性　**getdfree** は MS-DOS に特有の関数です。

関連項目　**getfat**, **getfatd**

getdisk

機能	カレントドライブを得ます。
形式	int getdisk(void)；
プロトタイプ	dir.h
解説	**getdisk** は，カレントドライブを得て，整数値（0＝A：，1＝B：，2＝C：，...）を返します。DOSのシステムコール0x19と同じです。
戻り値	**getdisk** は，カレントドライブ番号を返します。
可搬性	**getdisk** は MS-DOS に特有の関数です。
関連項目	**getcurdir, getcwd, setdisk**
例	**getcurdrive** を参照してください

getdta

機能　ディスク転送アドレスを得ます。

形式　char far ＊ getdta(void)；

プロトタイプ　dos.h

解説　**getdta** は，ディスク転送アドレス（DTA）の現在の設定を返します。
スモール，ミディアムのメモリモデルでは，セグメントは現在のデータセグメントであるとみなされます。C のみを使用していればそうなりますが，アセンブリ言語ルーチンを使用すると，ディスク転送アドレスはどんなハードウェアアドレスにも設定することができます。
コンパクト，ラージ，ヒュージのメモリモデルでは，**getdta** が返すアドレスは正しいハードウェアアドレスで，プログラムの外側に位置づけられていることもあります。

戻り値　**getdta** は，現在のディスク転送アドレスを指す far ポインタを返します。

可搬性　**getdta** は MS-DOS に特有の関数です。

関連項目　**setdta**

getenv

機能 環境から文字列を得ます。

形式 char * getenv(const char * *name*);

プロトタイプ stdlib.h

解説 **getenv**は，指定された環境変数の値を返します。環境変数名は大文字/小文字どちらで書いてもかまいませんが，等号（=）を含めてはいけません。指定された環境変数が見つからない場合は，**getenv**は空文字列を返します。

戻り値 成功した場合，**getenv**は*name*に結びつけられている値を返します。指定した*name*が環境の中で定義されていない場合は，**getenv**は空文字列を返します。

注意：環境変数を直接変更することはできません。環境の値を変更したい場合は，**putenv**関数を使ってください。

可搬性 **getenv**はUNIXシステムで使用でき，ANSI Cと互換性があります。

関連項目 *environ*(変数)，**getpsp**, **putenv**

例

```
#include <stdio.h>
#include <stdlib.h>

main()
{
   char *path, *dummy = NULL;

   path = getenv("PATH");
   dummy = getenv("DUMMY");

   printf("PATH = %s¥n", path);
   printf("old value of DUMMY: %s¥n",
          (dummy == NULL) ? "*none*" : dummy);
   putenv("DUMMY=TURBOC");
   dummy = getenv("DUMMY");
   printf("new value of DUMMY: %s¥n", dummy);
}
```

プログラム出力

```
PATH = C:¥BIN;C:¥BIN¥DOS;C:¥
old value of DUMMY: *none*
new value of DUMMY: TURBOC
```

getfat

機能　ファイルアロケーションテーブル情報を得る

形式　# include <dos.h>

void getfat(unsigned char *drive*, struct fatinfo * *dtable*);

プロトタイプ　dos.h

解説　**getfat** は，*drive* で指定されたドライブ(0=デフォルト，1=A，2=B，...)のファイルアロケーションテーブル（FAT）から情報を取り出します。*dtable* は，情報を格納する **fatinfo** 構造体を指します。

getfat が情報を格納する **fatinfo** 構造体は次のように定義されています。

```
struct fatinfo {
   char fi_sclus;         /* 1クラスタのセクタ数 */
   char fi_fatid;         /* FAT-id バイト */
   int  fi_nclus;         /* クラスタ数 */
   int  fi_bysec;         /* 1セクタのバイト数 */
};
```

戻り値　ありません。

可搬性　**getfat** は MS-DOS に特有の関数です。

関連項目　**getdfree**, **getfatd**

getfatd

機能　ファイルアロケーションテーブル情報を得ます。

形式　#include <dos.h>

void getfatd(struct fatinfo * *dtable*) ;

プロトタイプ　dos.h

解説　**getfatd**は，デフォルトドライブのファイルアロケーションテーブル（FAT）から情報を取り出します。*dtable*は，情報を格納する**fatinfo**構造体を指します。

getfatdが情報を格納する**fatinfo**構造体は次のように定義されています。

```
struct fatinfo {
   char fi_sclus;        /* 1クラスタのセクタ数 */
   char fi_fatid;        /* FAT-id バイト */
   int  fi_nclus;        /* クラスタ数 */
   int  fi_bysec;        /* 1セクタのバイト数 */
};
```

戻り値　ありません。

可搬性　**getfatd**はMS-DOSに特有の関数です。

関連項目　**getdfree**, **getfat**

getftime

機能　ファイルの日付と時刻を得ます。

形式　# include <io.h>

int getftime(int *handle*, struct ftime * *ftimep*)；

プロトタイプ　io.h

解説　**getftime** は，オープンされている *handle* に結びつけられているディスクファイルのファイル時刻・日付を取り出します。*ftimep* が指している **ftime** 構造体に，ファイル時刻・日付が格納されます。

ftime 構造体は次のように定義されています。

```
struct ftime {
   unsigned ft_tsec:  5;      /* 秒(2秒単位) */
   unsigned ft_min:   6;      /* 分          */
   unsigned ft_hour:  5;      /* 時          */
   unsigned ft_day:   5;      /* 日          */
   unsigned ft_month: 4;      /* 月          */
   unsigned ft_year:  7;      /* 年－1980    */
};
```

戻り値　成功した場合，**getftime** は0を返します。

エラーの場合は-1を返し，***errno*** に次のいずれかをセットします。

EINFNC	無効なファンクション番号
EBADF	ファイル番号が正しくない

可搬性　**getftime** は MS-DOS に特有の関数です。

関連項目　**open**, **setftime**

getpass

機能　パスワードを読みます。

形式　char * getpass(const char * *prompt*)；

プロトタイプ　conio.h

解説　**getpass** は，ヌル文字で終わる文字列 *prompt* をプロンプトとして表示し，システムコンソールからエコーバックなしでパスワードを読み込みます。ヌル文字で終わる最高8文字（ヌル文字は含めない）までの文字列を指すポインタを返します。

戻り値　戻り値は静的文字列へのポインタなので，呼び出されるごとに上書きされます。

可搬性　**getpass** は UNIX システムで使用できます.

関連項目　**getch**

getpsp

機能 プログラムセグメントプレフィックスを得ます。

形式 unsigned getpsp(void);

プロトタイプ dos.h

解説 **getpsp** は，DOS のシステムコール0x62を使って，プログラムセグメントプレフィクス（PSP）のセグメントアドレスを得ます。
この呼び出しは，MS-DOS 3.*x* にのみ存在します。MS-DOS 2.*x* では，スタートアップコードがセットするグローバル変数 ***_psp*** の値を，このかわりに使うことができます。

戻り値 **getpsp** は PSP のセグメントアドレスを返します。

可搬性 **getpsp** は MS-DOS 3.*x* に特有のものです。これより前のバージョンでは動作しません。

関連項目 **getenv**, ***_psp***(変数)

gets

機能 *stdin* から文字列を読み込みます。

形式 char * gets(char * *s*) ;

プロトタイプ stdio.h

解説 **gets** は，標準入力ストリーム *stdin* から，復帰文字で終わる文字列を読み込み，*s* に格納します。復帰文字は *s* の中ではヌル文字（¥0）に置き換えられます。

scanf とは違って **gets** では入力文字列中にホワイトスペース(空白，タブ)があってもかまいません。**get** は復帰文字に出会ったところで読み込みをやめ，それまでに読み込んだすべての文字を *s* にコピーします。

戻り値 成功した場合，**gets** は文字列引数 *s* を返します。ファイルエンドあるいはエラーの場合は NULL を返します。

可搬性 **gets** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **cgets, ferror, fgets, fputs, getc, puts**

例

```
#include <stdio.h>

main()
{
   char buff[133];

   puts("Enter a string: ");
   if (gets(buff) != NULL)
      printf("String = '%'¥n", buff);
}
```

gettime

機能　　システム時刻を得ます。

形式　　#include <dos.h>

void gettime(struct time * *timep*);

プロトタイプ　dos.h

解説　　**gettime** はシステムの現在の時刻を，*timep* が指す **time** 構造体に格納します。

time 構造体は次のように定義されています。

```
struct time {
    unsigned char ti_min;       /* 分      */
    unsigned char ti_hour;      /* 時      */
    unsigned char ti_hund;      /* 1/100秒 */
    unsigned char ti_sec;       /* 秒      */
};
```

戻り値　　ありません。

可搬性　　**gettime** は MS-DOS に特有の関数です。

関連項目　　**getdate, setdate, settime, stime, time**

getvect

機能　　割り込みベクタエントリを得ます。

形式　　void interrupt(* getvect(int *interruptno*))();

プロトタイプ　　dos.h

解説　　8086ファミリィのすべてのプロセッサは，割り込みベクタのセットを持っており，これには0〜255の番号がふられています。各ベクタの中の4バイト値は，割り込み関数が置かれている場所を示すアドレスです。
getvect は，*interruptno* で指定された割り込みベクタの値を読み出し，その値を，割り込み関数を指す（far）ポインタとして返します。*interruptno* の値は0〜255でなければなりません。

戻り値　　**getvect** は，*interruptno* で指定された割り込みベクタの現在の4バイト値を返します。

可搬性　　**getvect** は MS-DOS に特有の関数です。

関連項目　　**disable**, **enable**, **geninterrupt**, **setvect**

例

```
#include <stdio.h>
#include <dos.h>

/* getvect example */

void interrupt (*oldfunc)();
int looping = 1;

/* get_out - this is our new interrupt routine */

void interrupt get_out()
{
   /* restore to original interrupt routine */
   setvect(5,oldfunc);
   looping = 0;
}

/* capture_prtscr - installs a new interrupt for
   <Shift><PrtSc> */

/* arguments : func -- new interrupt function pointer */

void capture_prtscr(void interrupt  (*func)())
{
   /* save the old interrupt */
   oldfunc = getvect(5);
   /*install our interrupt handler */
   setvect(5,func);
}

void main ()
{

   puts("Press <Shift><Prt Sc> to terminate");
   /* capture the print screen interrupt */
   capture_prtscr(get_out);

   /* do noting */
   while (looping);

   puts("Success");
}
```

getverify

機能　DOSのベリファイフラグの状態を返します。

形式　int getverify(void)；

プロトタイプ　dos.h

解説　**getverify**は，ベリファイフラグの現在の状態を得ます。
ベリファイフラグはディスクへの出力を制御します。ベリファイがオフのときには書き込みの照合（ベリファイ）は行なわれず，ベリファイがオンのときはすべてのディスクへの書き込みにおいて，データが適切に書き出されたかどうかを確かめるために照合が行なわれます。

戻り値　**getverify**は，ベリファイフラグの現在の状態（0か1）を返します。

0 ＝ ベリファイフラグ オフ
1 ＝ ベリファイフラグ オン

可搬性　**getverify**はMS-DOSに特有の関数です。

関連項目　**setverify**

getw

機能　ストリームから整数を読み込みます。

形式

```
#include <stdio.h>
int getw(FILE *stream);
```

プロトタイプ　stdio.h

解説　**getw** は，入力ストリーム *stream* から次の整数を読み込みます。**getw** は，ファイルの中の境界（アラインメント）を考慮しません。
getw は，テキストモードでオープンされたストリームに対しては使用すべきではありません。

戻り値　**getw** は，入力ストリーム *stream* 上の次の整数を返します。ファイルエンドあるいはエラーの場合は EOF を返します。EOF は **getw** が通常の値として返すものなので，ファイルエンドあるいはエラーを検出するためには，**feof** や **ferror** を使う必要があります。

可搬性　**getw** は UNIX システムで使用できます。

関連項目　**putw**

gmtime

機能 日付および時刻をグリニッヂ時間に変換します。

形式 #include <time.h>

struct tm * gmtime(const time_t * timer);

プロトタイプ time.h

解説 **gmtime**は，**time**が返した値のアドレスを引数にとり，分解された時刻が含まれているtm型の構造体へのポインタを返します。**gmtime**は，GMT（グリニッジ標準時）への変換を行ないます。

long型のグローバル変数***timezone***は，GMTと地方標準時との差を秒単位で保持している必要があります（PST－太平洋標準時の場合には8×60×60)。グローバル変数***daylight***は，夏時間の期間中にのみ0以外の値をとります。

構造体**tm**は，time.hの中で以下のように定義されています。

```
struct tm {
   int tm_sec;
   int tm_min;
   int tm_hour;
   int tm_mday;
   int tm_mon;
   int tm_year;
   int tm_wday;
   int tm_yday;
   int tm_isdst;
};
```

これらは，24時間制の時刻，日（1～31），月（0～11），曜日（日曜が0），年の下2桁（年-1900），年内での日（0～365），夏時間かどうかのフラグ（夏時間の間は0以外の値）を示しています。

戻り値 **gmtime**は，時刻が分解されて入っている構造体を指すポインタを返します。この構造体は静的データで，呼び出しが行なわれるごとに上書きされます。

可搬性　gmtime は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　asctime, ctime, ftime, localtime, stime, time, tzset

例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

main()
{
   struct tm *timeptr;
   time_t     secsnow;

   timezone = 8 * 60 * 60;

   /* 1970-01-01 00:00:00 からの経過秒数を得る */
   time(&secsnow);

   /* GMT に変換 */
   timeptr = gmtime(&secsnow);
   printf("The date is %d-%d-19%02d¥n",
          (timeptr -> tm_mon) + 1, timeptr -> tm_mday,
          timeptr -> tm_year);
   printf("Greenwith Mean Time is %02d%02d:%02d¥n¥n",
          timeptr -> tm_hour, timeptr -> tm_min,
          timeptr -> tm_sec);
}
```

プログラム出力

```
The date is 2-2-1988
Greenwitch Mean Time is 20:44:36
```

harderr

機能　　ハードウェアエラーハンドラを確立します。

形式　　void harderr(int (* *handler*)());

プロトタイプ　dos.h

解説

harderr は，現在のプログラムに対するハードウェアエラーハンドラを定めます。このエラーハンドラは，割り込み0x24が起こったときに呼び出されます（割り込みにについては，MS-DOS プログラマーズリファレンスマニュアルを参照してください）。
そのような割り込みが起きたときに，*handler* が指す関数が呼び出されます。ハンドラ関数は次のような引数をともなって呼び出されます。

```
handler(int errval, int ax, int bp, int si);
```

errval は，DOS によって DI レジスタにセットされるエラーコードです。
ax, *bp*, *si* は，DOS が AX, BP, SI レジスタにセットする値です。

- *ax* は，ディスクあるいは他のデバイスのどちらでエラーが起きたのかを示します。*ax* が負でない値であればディスクエラーを示し，負であれば他のデバイスエラーを示します。ディスクエラーの場合，*ax* と 0x00FF の AND をとると，エラーの起きたドライブ番号(1=A, 2=B,...）を得ることができます。
- *bp* と *si* の2つで，エラーの起きたドライブのデバイスドライバヘッダを指します。

handler が指している関数が直接呼び出されることはありません。**harderr** が，その関数を呼び出す DOS 割り込みハンドラを確立します。
peek と **peekb** を使うと，このドライバヘッダからデバイスに関する情報を得ることができます。

ドライバヘッダは，**poke** あるいは **pokeb** によって変更することはできません。
ハンドラの中では1から0xC までのDOS コールを発行することができますが，これ以外のDOS コールはMS-DOS をこわすことになります。特に，すべてのC 標準I/O あるいはUNIX エミュレーションI/O を呼び出すことはできません。
ハンドラは，0（無視），1（再試行），2（異常終了）のいずれかの値を返さなければなりません。

戻り値　　ありません。

可搬性　　**harderr** はMS-DOS に特有の関数です。

関連項目　　**hardresume**, **handretn**, **peek**, **poke**

例

```
#include <stdio.h>
#include <dos.h>

#define DISPLAY_STRING   0x09
#define IGNORE 0
#define RETRY  1
#define ABORT  2

int handler(int errval, int ax, int bp, int si)
{
   char msg[25];  int drive;

   /* デバイスエラー */
   if (ax < 0)
   {
      /* DOS ファンクション 0 - 0x0C のみが使用可能 */
      bdosptr(DISPLAY_STRING, "device errors$", 0);
      hardretn(-1);     /* 呼び出しプログラムに戻る */
   }
   drive = (ax & 0x00FF);
   sprintf(msg, "disk error on drive %c$", 'A' + drive);
   bdosptr(DISPLAY_STRING, msg, 0);
   return(ABORT);       /* 呼び出しプログラムを異常終了させる */
}

main()
{
   harderr(handler);

   printf("Make sure there is no disk in drive A:¥n");
   printf("Press a key when ready...¥n");
   getch();

   printf("Attempting to access A:¥n");
   fopen("A:ANY.FIL","r");
}
```

プログラム出力

```
Make sure there is no disk in drive A:
Pless a key when ready...
Attempting to access A:
disk error on drive A
```

hardresume

機能　ハードウェアエラーハンドラ関数です。

形式　void hardresume(int *axret)*;

プロトタイプ　dos.h

解説　**harderr** によって確立されたエラーハンドラは，**hardresume** を呼び出して DOS へ制御を戻すことができます。**hardresume** の *axret*(結果コード)は，異常終了 (2)，再試行 (1)，無視 (0) のいずれかの値をもちます。異常終了は，DOS 割り込み0x23 (コントロールブレーク割り込み) の呼び出しによって行なわれます。
ハンドラは，0 (無視)，1 (再試行)，2 (異常終了) のいずれかの値を返さなければなりません。

戻り値　ありません。

可搬性　**hardresume** は MS-DOS に特有の関数です。

関連項目　**harderr, hardretn**

hardretn

機能　ハードウェアエラーハンドラ関数です。

形式　void hardretn(int *retn)；

プロトタイプ　dos.h

解説　**harderr** によって確立されたエラーハンドラは，**hardretn** を呼び出して，直接アプリケーションプログラムに戻ることができます。
ハンドラは，0（無視），1（再試行），2（異常終了）のいずれかの値を返さなければなりません。

戻り値　ありません。

可搬性　**hardretn** は MS-DOS に特有の関数です。

関連項目　**harderr**, **hardresume**

例　**harderr** を参照してください。

hypot

機能　直角三角形の斜辺の長さを計算します。

形式
```
#include <math.h>
double hypot(double x, double y);
```

プロトタイプ　math.h

解説　**hypot** は，次の式を満たすような z の値を計算します。

$z^2 = x^2 + y^2$ かつ $z >= 0$

これは，直角三角形の直角をはさむ2辺の長さ x と y がわかっている場合に，斜辺の長さを求めることと等しくなります。

戻り値　成功した場合，**hypot** は *z* を double で返します。エラーの場合(オーバーフローなど) は HUGE_VAL を返し，***errno*** に次の値をセットします。

ERANGE　　結果が範囲を越えた

hypot のエラー処理は，**matherr** 関数を使って変更することができます。

可搬性　**hypot** は UNIX システムで使用できます。

inport

機能　ハードウェアポートから1ワードを読み込みます。

形式　# include <dos.h>

int inport(int *portid*)；

プロトタイプ　dos.h

解説　**inport** は，*portid* に指定した入力ポートから1ワード読み込みます。

戻り値　**inport** は，読み込んだ値を返します。

可搬性　**inport** は80x86ファミリィに特有の関数です。

関連項目　**inportb**, **outport**, **outportb**

inportb

機能　ハードウェアポートから1バイト読み込みます。

形式　unsigned char inportb(int *portid*)；

プロトタイプ　dos.h

解説　**inportb** は，*portid* に指定した入力ポートから1バイトを読み込むマクロです。

inportb は，dos.h がインクルードされている場合は，インラインコードに展開されるマクロとして扱われます。

dos.h をインクルードしていない場合や，dos.h はインクルードしても **inportb** マクロを# undef している場合には関数として扱われます。

戻り値　**inportb** は，読み込んだ値を返します。

可搬性　**inportb** は80x86ファミリィに特有の関数です。

関連項目　**inport**, **outport**, **outportb**

int86

機能　8086ソフトウェア割り込みを行ないます。

形式　#include <dos.h>

int int86(int *intno*, union REGS ＊ *inregs*, union REGS ＊ *outregs*)；

プロトタイプ　dos.h

解説　**int86** は，引数 *intno* で指定された8086ソフトウェア割り込みを実行します。ソフトウェア割り込みを実行する前に，*inregs* 中の値を各レジスタにコピーします。

ソフトウェア割り込みが終わると，**int86** は，現在のレジスタの内容を *outregs* にコピーし，キャリーフラグのステータスを *outregs* の *x.cflag* フィールドにコピーし，さらに8086のフラグレジスタの値を *outregs* の *x.flags* フィールドにコピーします。

キャリーフラグがセットされている場合は，通常はエラーが起きたことを意味します。

inregs は，*outregs* と同じ構造体を指していてもかまいません。

戻り値　**int86** は，ソフトウェア割り込み終了時の AX レジスタの値を返します。キャリーフラグがセットされているとき (outregs->x.flag != 0) はエラーを意味し，***_doserrno*** にエラーコードがセットされます。

可搬性　**int86** は80x86アーキテクチャに特有の関数です。

関連項目　**bdos, bdosptr, geninterrupt, int86x, intdos, intdosx, intr**

例

```
#include <dos.h>

#define BIOS 0x18

/* カーソル位置の設定 (x桁, y行) */
void gotoxy(int x,int y)
{
   union REGS regs;

   regs.h.ah = 0x13;
   regs.x.dx = (((y - 1) * 80) + x - 1) * 2;
   int86(BIOS, &regs, &regs);
}
```

int86x

機能　8086ソフトウェア割り込みインターフェースです。

形式

```
# include <dos.h>
int int86x(int intno, union REGS * inregs, union REGS * outregs,
           struct SREGS * segregs);
```

プロトタイプ　dos.h

解説　**int86x**は，引数*intno*で指定された8086ソフトウェア割り込みを実行します。ソフトウェア割り込みを実行する前に，*inregs*中の値を各レジスタにコピーします。**int86x**はさらに，*segregs->x.ds*と*segregs->x.es*の値を対応するレジスタに置きます。これによって，farポインタを使うプログラムや，ラージデータモデルのプログラムで，ソフトウェア割り込みの実行中に使用するセグメントを指定できるようになります。
ソフトウェア割り込みから戻ると，**int86x**は，現在のレジスタの内容を*outregs*にコピーし，キャリーフラグのステータスを*outregs*の*x.cflag*フィールドにコピーし，8086のフラグレジスタの値を*outregs*の*x.flags*フィールドにコピーします。**int86x**はさらに，DSを回復し，*segregs->es*と*segregs->ds*に対応するセグメントレジスタの値をセットします。キャリーフラグがセットされる場合は，通常はエラーが起きたことを意味します。
int86xは，デフォルトのデータセグメントとは異なるDSの値や，ESの値を必要とする8086ソフトウェア割り込みを呼び出す場合に使うことができます。

戻り値　**int86x**は，ソフトウェア割り込み終了時のAXレジスタの値を返します。キャリーフラグがセットされているとき(outregs->x.flag != 0)はエラーを意味し，***_doserrno***にエラーコードがセットされます。

可搬性　**int86x**は80x86アーキテクチャに特有の関数です。

関連項目　**bdos, bdosptr, geninterrupt, intdos, intdosx, int86, intr, segread**

intdos

機能　DOS 割り込みインターフェースです。

形式　# include <dos.h>

int intdos(union REGS * *inregs*, union REGS * *outregs*) ;

プロトタイプ　dos.h

解説　**intdos** は，DOS 割り込み0x21を実行して，指定の DOS ファンクションを呼び出します。*inregs->h.al* の値によって，呼び出される DOS ファンクションが決まります。

intdos は，DOS ファンクションを実行する前に，*inregs* 中の値を各レジスタにコピーします。

0x21割り込みから戻ると，**intdos** は，現在のレジスタの内容を *outregs* にコピーし，キャリーフラグのステータスを *outregs* の *x.cflag* フィールドにコピーし，さらに8086のフラグレジスタの値を *outregs* の *x.flags* フィールドにコピーします。

キャリーフラグがセットされている場合は，通常はエラーが起こったことを意味します。

inregs は，*outregs* と同じ構造体を指していてもかまいません。

戻り値　**intdos** は，DOS ファンクションコール終了時の AX レジスタの値を返します。キャリーフラグがセットされているとき（outregs->x.cflag != 0）はエラーを意味し，***_doserrno*** にはエラーコードがセットされます。

可搬性　**intdos** は MS-DOS に特有の関数です。

関連項目　**bdos, geninterrupt, int86, int86x, intdosx, intr**

例

```
#include <stdio.h>
#include <dos.h>

/* ファイル名の削除: 成功すれば0, 失敗すると0以外を返す */

int delete_file(char near *filename)
{
   union REGS regs;
   int ret;

   regs.h.ah = 0x41;       /* ファイル削除 */
   regs.x.dx = (unsigned) filename;
   ret = intdos(&regs, &regs);
   /* キャリーフラグがセットされていればエラー */
   return(regs.x.cflag ? ret : 0);
}

main()
{
   int err;

   err = delete_file("NOTEXIST.$$$");
   printf("Able to delete NOTEXIST.$$$: %s¥n",
          (!err) ? "YES" : "NO");
}
```

プログラム出力

```
Able to delete NOTEXIST.$$$: NO
```

intdosx

機能 DOS 割り込みインターフェースです。

形式

```
#include <dos.h>
int intdosx(union REGS * inregs, union REGS * outregs,
            struct SREGS * segregs);
```

プロトタイプ dos.h

解説 **intdosx** は，DOS 割り込み0x21を実行して，指定の DOS ファンクションを呼び出します。*inregs*->*h.al* の値によって，呼び出される DOS ファンクションが決まります。**intdosx** は，DOS ファンクションを実行する前に，*inregs* 中の値を各レジスタにコピーし，*segregs*->*x.ds* と *segregs*->*x.es* の値を対応するレジスタに置きます。これによって，far ポインタを使うプログラムや，ラージデータモデルのプログラムで，ファンクションの実行中に使用するセグメントを指定できるようになります。

0x21の割り込みが終わると，**intdosx** は，現在のレジスタの内容を *outregs* にコピーし，システムのキャリーフラグのステータスを *outregs* の *x.cflag* フィールドにコピーし，さらに8086のフラグレジスタの値を *outregs* の *x.flags* フィールドにコピーします。**intdosx** はさらに，DS を回復し，*segregs*->*es* と *segregs*->*ds* に対応するセグメントレジスタの値をセットします。

キャリーフラグがセットされている場合は，通常はエラーが起こったことを意味します。

intdosx は，デフォルトのデータセグメントとは異なる DS の値や，ES の値を必要とする DOS ファンクションを呼び出す際に使うことができます。

inregs は，*outregs* と同じ構造体を指していてもかまいません。

戻り値 **intdosx** は，DOS ファンクションコール終了時の AX レジスタの値を返します。キャリーフラグがセットされているとき（*outregs*->x.cflag != 0）はエラーを意味し，**_doserrno** にはエラーコードがセットされます。

可搬性 **intdosx** は MS-DOS に特有の関数です。

関連項目　**bdos, geninterrupt, int86, int86x, intdos, intr, segread**

例

```
#include <stdio.h>
#include <dos.h>

/* ファイル名の削除: 成功すれば0, 失敗すると0以外を返す */

int delete_file(char far *filename)
{
   union  REGS regs;
   struct SREGS sregs;
   int    ret;

   regs.h.ah = 0x41;       /* ファイル削除 */
   regs.x.dx = FP_OFF(filename);
   sregs.ds  = FP_SEG(filename);
   ret = intdosx(&regs, &regs);
   /* キャリーフラグがセットされていればエラー */
   return (regs.x.cflag ? ret : 0);
}

main()
{
   int err;

   err = delete_file("NOTEXIST.$$$");
   printf("Able to delete NOTEXIST.$$$: %s¥n",
          (!err) ? "YES" : "NO");
}
```

プログラム出力

```
Able to delete NOTEXIST.$$$: NO
```

intr

機能　もう1つの8086ソフトウェア割り込みインターフェースです。

形式　# include <dos.h>
void intr(int *intno*, struct REGPACK ＊ *preg*)；

プロトタイプ　dos.h

解説　**intr** 関数は，ソフトウェア割り込みを実行するためのもう1つのインターフェースです。引数 *intno* で指定された8086ソフトウェア割り込みを生成します。
intr は，ソフトウェア割り込みの実行の前に，**REGPACK** 構造体＊*preg* の中の各値を，対応するレジスタにコピーします。ソフトウェア割り込みが終わると，**intr** は，現在のレジスタの値を，フラグも含めて *preg* の中にコピーします。
intr へ渡される引数は次の2つです。

intno　実行されるべき割り込み番号
preg　以下の値を持つ構造体のアドレス
(a) 呼び出し前は，入力するレジスタの値
(b) 割り込み後は，現在のレジスタの値

REGPACK 構造体は，dos.h の中で次のように定義されています。

```
struct REGPACK
{
   unsigned  r_ax, r_bx, r_cx, r_dx;
   unsigned  r_bp, r_si, r_di, r_ds, r_es, r_flags;
};
```

戻り値　**intr** は値を返しません。**REGPACK** 構造体＊*preg* に，割り込み後のレジスタの値が入っています。

可搬性　**intr** は80x86アーキテクチャに特有の関数です。

関連項目　　**geninterrupt, int86, int86x, intdos, intdosx**

ioctl

機能 I/O デバイスを制御します。

形式 int ioctl(int *handle*, int *func* [, void * *argdx*, int *argcx*]);

プロトタイプ io.h

解説 これは，DOS のシステムコール0x44(IOCTL)への直接のインターフェースです。具体的なファンクションは，*func* の値によって次のようになります。

0	デバイスコントロールデータを得る
1	デバイスコントロールデータをセットする（*argdx* の値）
2	*argdx* が指すアドレスに，*argcx* バイト読み込む
3	*argdx* が指すアドレスから，*argcx* バイト書き出す
4	2とほぼ同じ，ただし *handle* をドライブ番号として扱う(0＝デフォルト，1＝A など)
5	3とほぼ同じ，ただし *handle* をドライブ番号として扱う(0＝デフォルト，1＝A など)
6	入力ステータスを得る
7	出力ステータスを得る
8	メディアの交換性のテスト（DOS 3.x のみ）
11	リトライ回数の設定（DOS 3.x のみ）

ioctl は，デバイスチャネルに関する情報を得るために使用します。*func* が0，6，7では，通常のファイルも指定できます。それ以外の値の場合にはEINVAL エラーを返します。
システムコール0x44の引数や戻り値に関する詳細は，MS-DOS のプログラマーズリファレンスマニュアルを参照してください。
引数 *argdx* および *argcx* はオプションです。

ioctlは，特殊なファンクションに対するMS-DOS 2.0デバイスドライバへの直接的なインターフェースを提供します。この関数の厳密な動作は，各社のハードウェアやデバイスによって異なり，サポートしていない場合もあります。**ioctl**の使い方に関しては使用するシステムのBIOSに関するマニュアルを参照してください。

戻り値

*func*が0あるいは1の場合，戻り値はデバイス情報(IOCTL呼び出しのDXレジスタの値）です。

*func*が2～5の場合は，戻り値は実際に転送されたバイト数です。

*func*が6～7の場合，戻り値はデバイスステータスです。

エラーが起きた場合は-1を返し，***errno***に次のいずれかの値をセットします。

EINVAL	引数が正しくない
EBADF	ファイル番号が正しくない
EINVDAT	データが正しくない

可搬性

ioctlはUNIXシステムで使用できますが，引数や機能は異なります。UNIXバージョン7とシステムIIIとでは，**ioctl**の使用法が異なります。MS-DOSで使用した**ioctl**呼び出しをそのままUNIXで使用することはできません。また，MS-DOSマシン間の可搬性もほとんどありません。MS-DOS 3.*x*で，*func*の値として8と11の機能が追加されました。

例

```
#include <stdio.h>
#include <io.h>
#include <dir.h>

main()
{
   int stat;
   stat = ioctl(0,8,0,0);  /* ファンクション8で,メディアが */
                           /* 交換性可能なかどうかを調べる */
   printf("Drive %c %s changeable¥n",
          getdisk() + 'A', (stat == 0) ? "is" : "is not");
}
```

プログラム出力

```
Drive A is not changeable
```

isalnum

機能 文字が英字あるいは数字であるかどうかを判定します。

形式 # include <ctype.h>
int isalnum(int *c)*；

プロトタイプ ctype.h

解説 **isalnum** は，テーブルを参照して，ASCII コード（整数値）の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**isalnum** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値 **isalnum** は，*c* が英字（*A*～*Z*，*a*～*z*）または数字（0～9）の場合に，0以外の値を返します。

可搬性 **isalnum** は UNIX システムで使用できます。

isalpha

機能　　文字が英字であるかどうかを判定します。

形式　　# include <ctype.h>

int isalpha(int *c)*;

プロトタイプ　ctype.h

解説　　**isalpha** は，テーブルを参照して，ASCII コード(整数値)の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**isalpha** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値　　**isalpha** は，*c* が英字（*A*～*Z*，*a*～*z*）の場合に，0以外の値を返します。

可搬性　　**isalpha** は UNIX システムで使用できます。

isascii

機能　文字がASCII文字であるかどうかを判定します。

形式

```
#include <ctype.h>
int isascii(int c);
```

プロトタイプ　ctype.h

解説　**isascii**は，テーブルを参照して，ASCIIコード（整数値）の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。

isasciiは，すべての整数値に対して定義されています。

戻り値　**isascii**は，*c*が0～127(0x00～0x7F)の範囲内にある場合に，0以外の値を返します。

可搬性　**isascii**はUNIXシステムで使用できます。

isatty

機能　　デバイスタイプをチェックします。

形式　　int isatty(int *handle*) ;

プロトタイプ　io.h

解説　　**isatty** は，*handle* が，次に示すキャラクタデバイスのいずれかであるかどうかの判定を行ないます。

- ■端末
- ■コンソール
- ■プリンタ
- ■シリアルポート

戻り値　　*isatty* は，デバイスがキャラクタデバイスの場合に0以外の値を返し，それ以外は0を返します。

iscntrl

機能　文字が制御文字かどうかを判定します。

形式

```
#include <ctype.h>
int iscntrl(int c);
```

プロトタイプ　ctype.h

解説　**iscntrl** は，テーブルを参照して，ASCII コード（整数値）の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**iscntrl** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値　**iscntrl** は，*c* が削除文字（0x7F）あるいは通常の制御文字（0x00～0x1F）の場合に，0以外の値を返します。

可搬性　**iscntrl** は UNIX システムで使用でき，ANSI C と互換性があります。

isdigit

機能　文字が数字かどうかを判定します。

形式

```
#include <ctype.h>
int isdigit(int c);
```

プロトタイプ　ctype.h

解説　**isdigit** は，テーブルを参照して，ASCII コード（整数値）の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**isdigit** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値　**isdigit** は，*c* が数字（0～9）の場合に，0以外の値を返します。

可搬性　**isdigit** は UNIX システムで使用でき，ANSI C と互換性があります。

isgraph

機能　文字がプリント可能文字かどうかを判定します。

形式　# include <ctype.h>

int isgraph(int *c)*；

プロトタイプ　ctype.h

解説　**isgraph** は，テーブルを参照して，ASCII コード（整数値）の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**isgraph** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値　**isgraph** は，*c* が空白以外のプリント可能な文字の場合に，0以外の値を返します。

可搬性　**isgraph** は，UNIX システムで使用でき，ANSI C と互換性があります。

islower

機能　文字が英小文字かどうかを判定します。

形式　# include <ctype.h>

int islowr(int *c)*；

プロトタイプ　ctype.h

解説　**islower** は，テーブルを参照して，ASCII コード(整数値)の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**islower** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値　**islower** は，*c* が小文字（*a*～*z*）の場合に，0以外の値を返します。

可搬性　**islower** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

isprint

機能　文字がプリント可能文字かどうかを判定します。

形式

```
#include <ctype.h>
int isprint(int c);
```

プロトタイプ　ctype.h

解説　**isprint** は，テーブルを参照して，ASCII コード（整数値）の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**isprint** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値　**isprint** は，*c* がプリント可能な文字（0x20～0x7E）の場合に，0以外の値を返します。

可搬性　**isprint** は UNIX システムで使用でき，ANSI C と互換性があります。

ispunct

機能　文字が区切り文字かどうかを判定します。

形式

```
#include <ctype.h>
int ispunct(int c);
```

プロトタイプ　ctype.h

解説　**ispunct** は，テーブルを参照して，ASCII コード（整数値）の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**ispunct** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値　**ispunct** は，*c* が区切り文字の場合（**iscntrl** or **isspace**）に，0以外の値を返します。

可搬性　**ispunct** は，UNIX システムで使用でき，ANSI C と互換性があります。

isspace

機能　文字が空白文字かどうかを判定します。

形式
```
#include <ctype.h>
int isspace(int c);
```

プロトタイプ　ctype.h

解説　**isspace**は，テーブルを参照して，ASCIIコード(整数値)の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**isspace**は，**isascii**(*c*)が真の場合，または*c*がEOFの場合についてのみ定義されています。

戻り値　**isspace**は，*c*が，空白，タブ，復帰，改行，垂直タブ，フォームフィードの場合に，0以外の値を返します。

可搬性　**isspace**はUNIXシステムで使用でき，ANSI Cと互換性があります。K&Rでも定義されています。

isupper

機能　文字が英大文字かどうかを判定します。

形式　# include <ctype.h>

int isupper(int *c)*;

プロトタイプ　ctype.h

解説　**isupper** は，テーブルを参照して，ASCII コード（整数値）の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**isupper** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値　**isupper** は，*c* が英大文字（*A*～*Z*）の場合に，0以外の値を返します。

可搬性　**isupper** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

isxdigit

機能 文字が16進数の表記文字かどうかを判定します。

形式 # include <ctype.h>

int isxdight(int *c);

プロトタイプ ctype.h

解説 **isxdigit** は，テーブルを参照して，ASCII コード(整数値)の分類を行なうマクロです。判定の結果が真であれば0以外の数値を，偽であれば0を返します。**isxdigit** は，**isascii**(*c*)が真の場合，または *c* が EOF の場合についてのみ定義されています。

戻り値 **isxdigit** は，*c* が16進数の表記文字（0～9，*A*～*F*，*a*～*f*）の場合に，0以外の値を返します。

可搬性 **isxdigit** は，UNIX システムで使用でき，ANSI C と互換性があります。

itoa

機能　整数を文字列に変換します。

形式　char ＊ itoa(int *value*, char ＊ *string*, int *radix*)；

プロトタイプ　stdlib.h

解説　**itoa** は，整数値 *value* の値をヌルで終わる文字列に変換し，結果を *string* に代入します。
radix には，*value* を変換する際に使用する基底を指定します。*radix* は，2〜36の値でなければなりません。*value* が負で *radix* が10であると，*string* の最初の文字は負符号（-）になります。

注意：*string* 用に確保する領域は，返される文字列を格納するのに充分な大きさでなければなりません。**itoa** の最大長は，最後のヌル文字も含めて17文字です。

戻り値　**itoa** は，文字列へのポインタを返します。エラーの戻り値はありません。

関連項目　**ltoa, ultoa**

kbhit

機能　キーボードが押されたかどうかをチェックします。

形式　int kbhit(void)；

プロトタイプ　conio.h

解説　**kbhit** は，現在までにキーボードが押されたかどうかのチェックを行ないます。押されたキーは，**getch** または **getche** で得ることができます。

戻り値　キーボードが押されていれば，**kbhit** は0以外の値を返し，押されていなければ0を返します。

関連項目　**getch**, **getche**

keep

機能　プログラムを常駐させたまま終了します。

形式　void keep(unsigned char *status*, unsigned *size*)；

プロトタイプ　dos.h

解説　**keep** は，*status* を終了ステータスとして DOS に戻ります。このとき，プログラムはメモリに常駐したままになります。プログラムの大きさは *size* パラグラフにセットされ，残りのメモリは開放されます。

keep は，TSR プログラムをインストールするときに使用することができます。**keep** は，DOS ファンクション0x31を使用します。

戻り値　ありません。

可搬性　**KEEP** は MS-DOS に特有の関数です。

関連項目　**abort**, **exit**

labs

機能　　long の絶対値を与えます。

形式

```
#include <math.h>
long int labs(long int x);
```

プロトタイプ　math.h, stdlib.h

解説　　**labs** は，long 型の引数 *x* の絶対値を返します。

戻り値　　**labs** は，*x* の絶対値を返します。エラーの戻り値はありません。

可搬性　　**labs** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**abs, cabs, fabs**

ldexp

機能 $x \times 2^{exp}$を計算します。

形式

```
#include <math.h>
double ldexp(double x, int exp);
```

プロトタイプ math.h

解説 **ldexp** は，$x \times 2^{exp}$を double で計算します。

戻り値 **ldexp** は，$x \times 2^{exp}$の値を返します。

ldexp のエラー処理は，**matherr** を通して変更することができます。

可搬性 **ldexp** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **exp**, **frexp**, **modf**

ldiv

機能 2つの long の除算を行ない，商と剰余を返します。

形式 # include <stdlib.h>

ldiv_t ldiv(long int *numer*, long int *denom*)；

プロトタイプ stdlib.h

解説 **ldiv** は，2つの long の除算を行なって，商と剰余を *ldiv_t* 型で返します。*numer* が被除数，*denom* が除数です。*ldiv_t* 型は long からなる構造体で，stdlib.h 中で次のように typedef 宣言されています。

```
typedef struct {
   long  quot;        /* 商   */
   long  rem;         /* 剰余 */
} ldiv_t;
```

戻り値 **ldiv** は，*quot*（商）と *rem*（剰余）を要素に持つ構造体を返します。

可搬性 **ldiv** は ANSI C と互換性があります。

関連項目 **div**

例

```
#include <stdlib.h>
ldiv_t lx;

main()
{
   lx = ldiv(100000L, 30000L);
   printf("100000 div 30000 = %ld remainder %ld¥n",
          lx.quot, lxrem);
}
```

lfind

機能　線形探索を行ないます。

形式

```
#include <stdlib.h>
void * lfind(const void * key, const void * base,
             size_t * num, size_t width,
             int (* fcmp)(const void *, const void *));
```

プロトタイプ　stdlib.h

解説　**lfind**は，シーケンシャルレコードの配列内で，*key*の値によって線形探索を行ないます。探索には，ユーザ定義の比較ルーチン（*fcmp*）が使われます。

探索の対象となる配列は，レコード長が*width*バイト，レコード数が* *num*で，*base*が指すアドレスから始まるものとして定義されます。

*fcmp*は，ユーザが定義する比較ルーチンを指すポインタです。比較ルーチンは，*elem1*と*elem2*の2つの引数をとります。2つの引数はそれぞれ比較される値を指します。比較関数は，2つの引数が指す項目（* *elem1*，* *elem2*）を比較し，その結果にしたがって整数値を返します。

* *fcmp*は，次のような値を返さなければなりません。

- ■* *elem1* != * *elem2* の場合，0以外
- ■* *elem1* == * *elem2* の場合，0

通常は，*elem1*は引数*key*で，*elem2*は検索の対象となるテーブル内の項目を指します。*fcmp*が検索キーやテーブルの項目をどのように解釈するかは自由であり，ユーザが必要とする方法で定義することができます。

戻り値　**lfind**は，配列内で検索キーと一致した最初の項目のアドレスを返します。一致する項目がなかった場合にはNULLを返します。

関連項目　**bsearch**, **lsearch**

localtime

機能　日付および時刻を構造体に変換します。

形式　# include <time.h>

struct tm * localtime(const time_t * *timer*)

プロトタイプ　time.h

解説　**localtime**は，**time**が返した値のアドレスを引数にとり，分解された時刻が入っている構造体を指すポインタを返します。**localtime**は，地方時間帯の時刻を返し，夏時間の場合にはこれも考慮します。

long型のグローバル変数 ***timezone*** は，GMTとその地方の標準時の差を秒単位で保持しています（PST－太平洋標準時の場合には8×60×60）。グローバル変数 ***daylight*** は，夏時間の期間中にのみ0以外の値をとるようになっています。

構造体**tm**は，time.hの中で以下のように定義されています。

```
struct tm {
   int tm_sec;
   int tm_min;
   int tm_hour;
   int tm_mday;
   int tm_mon;
   int tm_year;
   int tm_wday;
   int tm_yday;
   int tm_isdst;
};
```

これらは，24時間制の時刻，日（1～31），月（0～11），曜日（日曜が0），年の下2桁（年-1900），年内での日（0～365），夏時間かどうかのフラグ（夏時間の間は0以外の値）を示しています。

戻り値　**localtime**は，時刻が分解されて入っている構造体を指すポインタを返します。この構造体は静的データで，呼び出しが行なわれるごとに上書きされます。

可搬性　**localtime** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**asctime, ctime, ftime, gmtime, stime, time, tzset**

例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

main()
{
   struct tm *timeptr;
   time_t     secsnow;

   timezone = 8 * 60 * 60;
   time(&secsnow);
   timeptr = localtime(&secsnow);
   printf("The date is %d-%d-19%02d¥n",
          ((timeptr -> tm_mon) + 1), time -> tm_mday,
          timeptr -> tm_year);
   printf("Local time is %02d:%02d:%02d¥n",
          timeptr -> hour, timeptr -> tm_min,
          timeptr -> tm_sec);
}
```

プログラム出力

```
The date is 2-2-88
Local time is 12:44:36
```

lock

機能 ファイルシェアリングのロックをセットします。

形式 int lock(int *handle*, long *offset*, long *length*);

プロトタイプ io.h

解説 **lock** は，DOS 3.*x* のファイルシェアリング機構へのインターフェースを提供します。

lock は，ファイルの任意の重なりのない領域に対して実行することができます。ロックされた領域に対して読み書きするプログラムは，3回までその操作を再試行します。3回とも失敗した場合は，その呼び出しはエラーで終わることになります。

戻り値 **lock** は，成功した場合は0を返し，エラーの場合は-1を返します。

可搬性 **lock** は MS-DOS 3.0に特有の関数で，それより前のバージョンでは動作しません。

関連項目 **open, sopen, unlock**

log

機能　　自然対数を計算します。

形式
```
#include <math.h>
double log(double x);
```

プロトタイプ　math.h

解説　　**log** は，*x* の自然対数を計算します。

戻り値　　**log** は，成功した場合は計算結果を返します。
引数 *x* が0以下であった場合は，**log** は HUGE_VAL を返し，***errno*** に次の値をセットします。

EDOM　　定義域エラー

log のエラー処理は，**matherr** 関数を通して変更することができます。

可搬性　　**log** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**exp, log10, sqrt**

log10

機能　常用対数を計算します。

形式　# include <math.h>
double log10(double *x)*；

プロトタイプ　math.h

解説　**log10** は，*x* の常用対数を計算します。

戻り値　**log10** は，成功した場合は計算結果を返します。
引数 *x* が0以下であった場合は，**log10** は HUGE_VAL を返し，***errno*** に次の値をセットします。

EDOM　　定義域エラー

log10 のエラー処理は，**matherr** 関数を通して変更することができます。

可搬性　**log10** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**exp**, **log**

longjmp

機能 ローカルでない goto を行ないます。

形式 #include <setjmp.h>

void longjmp(jmp_buf *jmpb*, int *retval*);

プロトタイプ setjmp.h

解説 **longjmp** を呼び出すと，*jmpb* を引数とする **setjmp** の前回の呼び出しでとらえられたタスクの状態が回復されます。そして，**setjmp** が値 *retval* を返したかのようにリターンします。

タスクの状態とは以下のものをいいます。

■すべてのセグメントレジスタ（CS，DS，ES，SS）

■レジスタ変数（SI，DI）

■スタックポインタ（SP）

■フレームポインタ（FP）

■フラグ

タスクの状態は，コルーチンを実現するために **setjmp** と **longjmp** が使用されるのに充分なものです。

setjmp は，**longjmp** より前に呼び出しておかなければなりません。**setjmp** を呼んで *jmpb* を設定したルーチンは，**longjmp** が呼び出されるまではアクティブなままでなければならず，リターンすることはできません。もし，そうしてしまった場合は，結果は予測できません。

これらのルーチンは，プログラムの低レベルサブルーチンにおいて起こるエラーや例外を処理する際に便利なものです。

longjmp は値0を返すことはできません。*retval* に0を入れて渡すと，**longjmp** はこれを1に置き換えます。

戻り値 ありません。

可搬性　　**longjump** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**setjmp**，**signal**

例

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf jumper;

main()
{
   int value;

   value = setjmp(jumper);
   if (value != 0)
   {
      printf("Longjmp with value %d¥n", value);
      exit(value);
   }
   printf("About to call subroutine ... ¥n");
   subroutine();
}

subroutine()
{
   longjmp(jumper,1);
}
```

プログラム出力

```
About to call subroutine ...
Longjmp with value 1
```

_lrotl

機能 unsigned long の値を左へローテートします。

形式 unsigned long _lrotl(unsigned long *val*, int *count*);

プロトタイプ stdlib.h

解説 **_lrotl** は，与えられた値 *val* を，指定されたビット数 *count* だけ左へローテートします。*val* は unsigned long です。

戻り値 **_lrotl** は，*val* を *count* ビット左へローテートした値を返します。

関連項目 **_lrotr**, **_rotl**

例 **_rotl** を参照してください。

_lrotr

機能　unsigned long の値を右へローテートします。

形式　unsigned long _lrotr(unsigned long *val*, int *count*)；

プロトタイプ　stdlib.h

解説　**_lrotr** は，与えられた値 *val* を指定されたビット数 *count* だけ右へローテートします。*val* は unsigned long です。

戻り値　**_lrotr** は，*val* を *count* ビット右へローテートした値を返します。

関連項目　**_lrotl**, **_rotr**

例　**_rotl** を参照してください。

lsearch

機能　線形探索を行ないます。

形式

```
#include <stdlib.h>
void * lsearch(const void * key, void * base,
               size_t * num, size_t width,
               int (* fcmp)(const void *, const void *));
```

プロトタイプ　stdlib.h

解説

lsearch は，テーブル中の情報を検索します。**lsearch** は線型探索を行なうので，呼び出しの前にテーブルがソートされている必要はありません。*key* が指している項目がテーブルの中に存在しない場合，**lsearch** はその項目をテーブルに追加します。

- *base* は，検索するテーブルの基底（0番目の要素）を指します。
- *num* は，テーブル内の要素の個数を示す数値を指します。
- *width* は，1つの項目の大きさ（バイト数）を示します。
- *key* は，検索する項目（検索キー）を指します。

fcmp は，ユーザが定義する比較ルーチンを指すポインタです。比較ルーチンは2つの引数（*elem1*，*elem2*）をとり，この引数はそれぞれ比較される値を指します。比較関数は，2つの引数が指す項目（* *elem1*，* *elem2*）を比較し，その結果にしたがって整数値を返します。通常は，*elem1* は引数 *key* で，*elem2* は検索の対象となるテーブル内の項目を指します。*fcmp* は，次のような値を返す必要があります。

- * *elem1* == * *elem2* の場合，0
- * *elem1* != * *elem2* の場合，0以外の値

fcmp が検索キーやテーブルの項目をどのように解釈するかは自由であり，ユーザが必要とする方法で定義することができます。

戻り値　　**lsearch** は，テーブル内で検索キーと一致した最初の項目のアドレスを返します。

可搬性　　**lsearch** は UNIX システムで使用できます。

関連項目　　**bsearch, lfind, qsort**

例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>   /* strcmp の宣言のため */

/* カラーの番号を初期化 */
char *colors[10] = { "Red", "Blue", "Green" };
int ncolors = 3;

int colorscmp(char **arg1, char **arg2)
{
   return(strcmp(*arg1, *arg2));
}

int addelem(char *color)
{
   int oldn = ncolors;
   lsearch(&color, colors, (size_t *) &ncolors,
           sizeof(char *), colorscmp);
   return (ncolors == oldn);
}

main()
{
   int i;
   char *key = "Purple";

   if (addelem(key))
      printf("%s already in colors table¥n", key);
   else
      printf("%s added to colors table, now %d colors¥n",
             key, ncolors);
   printf("The colors:¥n");
   for (i = 0; i < ncolors; i++)
      printf("%s¥n", colors[i]);
}
```

プログラム出力

```
purple added to colors table,
now 4 colors
```

lseek

機能　ファイルポインタを移動します。

形式

```
#include <io.h>
long lseek(int handle, long offset, int fromwhere);
```

プロトタイプ　io.h

解説　**lseek** は，*handle* に結びつけられているファイルポインタを，*fromwhere* で指定された位置から *offset* バイト離れた位置に移動します。*fromwhere* は，0，1，2のいずれかでなければなりません。*fromwhere* には，stdio.h の中で定義されている次の3つのシンボリック定数を用いると便利です。

fromwhere		**ファイルの位置**
SEEK_SET	(0)	ファイルの先頭
SEEK_CUR	(1)	ファイルポインタの現在位置
SEEK_END	(2)	ファイルの終わり

戻り値　**lseek** は，ポインタの新しい位置のオフセットを，ファイルの先頭からのバイト数で返します。エラーの場合は-1L を返し，***errno*** は次のいずれかにセットされます。

EBADF	ファイル番号が正しくない
EINVAL	正しくない引数

シークが行なえないデバイス（端末やプリンタ）に対しては，戻り値は未定義です。

可搬性　**lseek** は UNIX システムで使用できます。

関連項目　　**filelength, fseek, ftell, sopen, _write, write**

ltoa

機能　long 型を文字列に変換します。

形式

```
#include <stdlib.h>
char * ltoa(long value, char * string, int radix);
```

プロトタイプ　stdlib.h

解説　**ltoa** は，*value* の値をヌルで終わる文字列に変換し，結果を *string* に格納します。

radix には，*value* を変換する際に使用する基底を指定します。*radix* は，2～36の値でなければなりません。*value* が負で *radix* が10であると，文字列の最初の文字は負符号（-）になります。

注意：*string* 用に確保する領域は，返される文字列を格納するのに充分な大きさでなければなりません。**ltoa** が返す最大文字数は，最後のヌル文字（¥0）も含めて33文字です。

戻り値　3つの関数とも文字列へのポインタを返します。エラーの戻り値はありません。

関連項目　**itoa**, **ultoa**

malloc

機能　メインメモリを割り当てます。

形式

```
# include <stdlib.h> or # include <alloc.h>
void * malloc(size_t size) ;
```

プロトタイプ　stdlib.h, alloc.h

解説　**malloc**は，Cのメモリヒープから*size*バイトのブロックを確保します。これによって，プログラムは，必要なときに必要な量だけメモリを確保することができます。

ヒープは，可変サイズのメモリブロックの動的割り当てに使用されます。木構造やリストといった多くのデータ構造は，通常ヒープメモリに割り当てられます。

スモールデータモデルでは，データセグメントの終わりからプログラムスタックの先頭までのすべての領域をヒープとして使用することができます。ただし，スタックトップの直前の256バイトは，アプリケーション用のスタックの拡張領域とDOSが使用する領域として確保されているため除外されます。

ラージデータモデルでは，プログラムスタックを越えてメモリの物理的な最終位置までのすべての領域がヒープとして使用可能です。

戻り値　**malloc**は，*size*に指定した長さのメモリブロックを指すポインタを返します。要求したブロックを確保するだけのメモリが充分ない場合，**malloc**はNULLを返し，ブロックの内容はそのままになります。

引数*size*＝＝0の場合にはNULLを返します。

可搬性　**malloc**はUNIXシステムで使用でき，ANSI Cと互換性があります。

関連項目　**allocmem, calloc, coreleft, farcalloc, farmalloc, free, realloc**

例

```
#include <stdio.h>
#include <stdlib.h>

typedef struct  {
   /* ... */
} OBJECT;

OBJECT *NewObject()
{
   return ((OBJECT *) malloc(sizeof(OBJECT)));
}

void FreeObject(OBJECT *obj)
{
   free(obj);
}

main()
{
   OBJECT *obj;

   obj = NewObject();
   if (obj == NULL) {
      printf("failed to create a new object¥n");
      exit(1);
   }
   /* ... */
   free(obj);
}
```

_matherr

機能　浮動小数点エラーを処理します。

形式

```
# include <math.h>
double _matherr(_mexcep why, char * fun,
                double * arg1p, double * arg2p, double retval);
```

プロトタイプ　math.h

解説

_matherrは，すべての数学ライブラリ関数内のエラー処理において中心的に機能する関数です。**_matherr**は，**matherr**を呼び出して，**matherr**が返す値を処理します。**_matherr**は，ユーザプログラムから直接呼び出すことはできません。数学ライブラリのエラー処理は，**matherr**関数を置き換えることによってカスタマイズすることができます。

数学ライブラリルーチンのいずれかでエラーが起こると，いくつかの引数とともに**_matherr**が呼び出されます。**_matherr**は次の4つの処理を行ないます。

- 渡された引数で**exception**構造体を埋めます。
- **exception**構造体を指すポインタ*e*を引数として**matherr**を呼び出し，**matherr**がそのエラーを解決できるかどうかを調べます。
- **matherr**からの戻り値を調べて，次の処理を行ないます。
  matherrが0を返した（**matherr**がそのエラーを解決できなかった）場合，**_matherr**は***errno***をセットしてエラーメッセージを出力します。
  matherrが0でない値を返した（**matherr**がそのエラーを解決できた）場合は，**_matherr**は何もしません。
- もとの呼び出し側へ，*e->retval*を返します。**matherr**は，*e->retval*を，もとの呼び出し側へ返したい値に変更できることに注意してください。

_matherr が ***errno*** をセットする場合（**matherr** からの戻り値が0），起こったエラーの種類（**exception** 構造体の *type* フィールド）を ***errno***（値はEDOM か ERANGE）に入れることになります。

戻り値 **_matherr** は，*e->retval* の値を返します。この値は，初めは **_matherr** に渡された引数 *retval* の値であり，**matherr** によって変更されることもあります。

数学関数の結果が MAXDOUBLE より大きかった場合，デフォルトでは *retval* は **_matherr** に渡される前に，HUGE_VAL に適切な符号をつけたものにセットされます。数学関数の結果が MINDOUBLE より小さかった場合は，*retval* は0にセットされ，**_matherr** に渡されます。どちらの場合でも，**matherr** が *e->retval* の値を変換しない場合は，**_matherr** が ***errno*** に ERANGE（結果が範囲外）をセットします。

関連項目 **matherr**

matherr

機能　ユーザが変更可能な数学エラーハンドラです。

形式　# include <math.h>

int matherr(struct exception * *e*)；

プロトタイプ　math.h

解説　**matherr** は，数学ライブラリが生成したエラーを処理するために _**matherr** ルーチンによって呼び出されます。

matherr は，ユーザ独自の数学エラー処理ルーチンが必要な場合に，これを置き換えて使用するフックとして提供されています（後で示す例を参照してください)。

matherr は，数学関数が引き起こした定義域エラーおよび値域エラーをトラップするのに便利です。ゼロによる除算などの浮動小数点の例外をトラップすることはできません。こうしたエラーのトラップについては **signal** を参照してください。

matherr を，独自のエラー処理ルーチン（あるタイプのエラーを捉えて解決する）として定義することができます。修正した **matherr** は，エラーを解決できなかった場合には0，解決できた場合には0でない値を返さなければなりません。**matherr** が0でない値を返したときは，エラーメッセージはプリントされず，***errno*** は変更されません。

exception 構造体は，math.h の中で次のように定義されています。

```
struct exception {
   int    type;
   char   *name;
   double arg1, arg2, retval;
};
```

exception 構造体のメンバは次のようになっています。

メンバ	意味
type	起きた数学エラーのタイプ。typedef 名 _*mexcep*（後述）の中で定義された enum 型の値。
name	エラーを起こした数学ライブラリ関数の名前を持つ文字列を指すポインタ。
arg1, *arg2*	エラーを起こした引数。関数へ渡される引数が1つだけの場合は *arg1* に入れる。
retval	そのエラーのデフォルトの戻り値。この値は変更することができる。

typedef 名 _*mexcep*（これも math.h の中で定義されています）は，次に示すそれぞれの数学エラーに対応するシンボリック定数を定義しています。

シンボリック定数	数学エラー
DOMAIN	関数の領域に引数が入っていない。例：**log**(-1)。
SING	引数が特異性を引き起こす。例：**pow**(0,-2)。
OVERFLOW	引数のため，関数の結果が MAXDOUBLE を越える。例：**exp**(1000)。
UNDERFLOW	引数のため，関数の結果が MINDOUBLE を下回る。例：**exp**(-1000)。
TLOSS	引数のため，関数の結果の有効数字が減る。例：**sin**(10e70)。

シンボリック定数 MAXDOUBLE, MINDOUBLE は values.h の中で定義されています。

_matherr を変更することはできないことに注意してください。**matherr** は，多くのＣランタイムライブラリに含まれており，**matherr** を使用すれば可搬性を高めることができます。

UNIXスタイルの**matherr**のデフォルトの動作（メッセージをプリントして終了する）は，ANSIとの互換性はありません。UNIXスタイルを望むのであれば，Turbo Cのマスターディスクに含まれているmatherr.cを使用してください。

戻り値

matherrのデフォルトの戻り値は，エラーがUNDERFLOWあるいはTLOSSの場合には1，それ以外の場合には0です。**matherr**は，*e->retval*を変更することもできます。変更した値は，**_matherr**を経由してもとの呼び出し側に返されることになります。

matherrが0を返した（エラーを解決することができなかった）場合は，**_matherr**が***errno***をセットしてエラーメッセージを出力します（**_matherr**を参照してください）。

matherrが0でない値を返した（エラーを解決することができた）場合は，***errno***はセットされず，メッセージも出力されません。

可搬性

matherrは，多くのCコンパイラで使用できますが，ANSIとの互換性はありません。

メッセージをプリントして終了するUNIXスタイルの**matherr**は，Turbo Cのマスターディスクに含まれているMATHERR.Cで提供されています。

関連項目

_matherr

例

```
/* これはユーザ定義のmatherr関数で，sqrtに負の引数が渡された場合に
   sqrtがその値を処理する前に負でない数値に変換します．              */

#include <math.h>
#include <string.h>

int matherr(struct exception *a);
{
   if (a -> type == DOMAIN)
   {
      if(strcmp(a -> name, "sqrt") == 0)
      {
         a -> retval = sqrt (-(a -> arg1));
         return (1);
      }
   }
   return (0);
}
```

max

機能 2つの値の大きい方を返します。

形式 # include <stdlib.h>
(type) max(*a*, *b*);

プロトタイプ stdlib.h

解説 **max** は，2つの値を比較して大きい方を返すマクロです。2つの引数と関数の型は，同じものとして宣言されなければなりません。

戻り値 **max** は，大きい方の値を返します。

関連項目 **min**

例

```
#include <stdlib.h>

main()
{
   int x = 5;
   int y = 6;
   int z;

   z = (int) max(x, y);
   printf("The larger number is %d¥n", z);
}
```

プログラム出力

```
The larger number is 6
```

memccpy

機能 メモリブロックをコピーします。

形式 # include <mem.h>

void * memccpy(void * *dest*, const void * *src*, int *c*, size_t *n*);

プロトタイプ string.h, mem.h

解説 **memccpy** は，*src* から *dest* へ *n* バイトのブロックをコピーをします。コピーは，次の条件のいずれかが満たされると終了します。

■文字 *c* が *dest* に初めてコピーされた。
■ *n* バイトが *dest* にコピーされた。

戻り値 **memccpy** は，*c* がコピーされた場合は *dest* 中の *c* の次のバイトを指すポインタを返し，そうでない場合は NULL を返します。

可搬性 **memccpy** は UNIX システム V で使用できます。

関連項目 **memcpy**, **memmove**, **memset**

memchr

機能 メモリブロック中で文字を検索します。

形式
```
#include <mem.h>
void * memchr(const void * s, int c, size_t n);
```

プロトタイプ string.h, mem.h

解説 **memchr** は，*s* が指しているメモリブロックの最初の *n* バイトの中で，文字 *c* を探します。

戻り値 **memchr** は，*s* の中で最初に *c* が見つかった位置を指すポインタを返します。見つからなければ NULL を返します。

可搬性 **memchr** は UNIX システム V で使用でき，ANSI C と互換性があります。

memcmp

機能　2つのメモリブロックを比較します。

形式　# include <mem.h>

void * memcmp(const void * *s1*, const void * *s2*, size_t *n*) ;

プロトタイプ　string.h, mem.h

解説　**memcmp** は，*s1* と *s2* が指す2つのブロックの先頭から *n* バイトを，各バイトを unsigned char として比較します。

戻り値　**memcmp** は，次のような値を返します。

<0	*s1* が *s2* より小さい場合
=0	*s1* と *s2* が等しい場合
<0	*s1* が *s2* より大きい場合

各バイトは unsigned char として比較されるので，次の式は正の値を返します。

```
memcmp("¥xFF","¥x7F",1)
```

可搬性　**memcmp** は UNIX システム V で使用でき，ANSI C と互換性があります。

関連項目　**memicmp**

memcpy

機能　メモリブロックをコピーします。

形式　# include <mem.h>

void ＊ memcpy (void ＊ *dest*, const void ＊ *src*, size_t *n*)；

プロトタイプ　string.h, mem.h

解説　**memcpy** は，*src* から *dest* へ *n* バイトのブロックをコピーします。*src* と *dest* がオーバーラップしている場合の動作は未定義です。

戻り値　**memcpy** は *dest* を返します。

可搬性　**memcpy** は UNIX システム V で使用でき，ANSI C と互換性があります。

関連項目　**memccpy**, **memmove**, **memset**, **movedata**, **movemem**

memicmp

機能　2つのメモリブロックを文字ケースを無視して比較します。

形式　# include 〈mem.h〉

int　memicmp(const void ＊ *s1*, const void ＊ *s2*, size_t *n*)；

プロトタイプ　string.h, mem.h

解説　**memicmp** は，*s1* と *s2* が指す2つのブロックの先頭から *n* バイトを，文字ケース（大文字か小文字か）を無視して比較します。

戻り値　**memicmp** は，次のような値を返します。

〈0	*s1* が *s2* より小さい
＝0	*s1* と *s2* が等しい
〈0	*s1* が *s2* より大きい

可搬性　**memicmp** は UNIX システム V で使用できます。

関連項目　**memcmp**

memmove

機能 メモリ配列を操作する

形式 # include <mem.h>

void * memmove(void * *dest*, const void * *src*, size_t *n*) ;

プロトタイプ string.h,mem.h

解説 **memmove** は，*src* から *dest* へ *n* バイトのブロックをコピーします。*src* と *dest* でオーバーラップしている場合でも，オーバーラップ部分は正しくコピーされます。

戻り値 **memmove** は *dest* を返します。

可搬性 **memmove** は UNIX システム V で使用でき，ANSI C と互換性があります。

関連項目 **memccpy, memcpy, movmem**

memset

機能 メモリブロックに指定された文字をセットします。

形式 # include <mem.h>

void * memset(void * *s*, int *c*, size_t *n*);

プロトタイプ string.h, mem.h

解説 **memset** は, *s* が指すブロックの先頭から *n* バイトに, 文字 *c* をセットします。

戻り値 **memset** は *s* を返します。

可搬性 **memset** は UNIX システム V で使用でき, ANSI C と互換性があります。

関連項目 **memccpy**, **memcpy**, **setmem**

min

機能　2つの値の小さい方を返します。

形式　# include <stdlib.h>
(type) min(*a*, *b*) ;

プロトタイプ　stdlib.h

解説　**min** は，2つの値を比較して小さい方を返すマクロです。2つの引数と関数の型は，同じものとして宣言されなければなりません。

戻り値　**min** は，小さい方の値を返します。

関連項目　**max**

例

```
#include <stdlib.h>

main()
{
   int x = 5;
   int y = 6;
   int z;

   z = (int) min(x, y);
   printf("The smaller number is %d¥n", z);
}
```

プログラム出力

```
The smaller number is 5
```

mkdir

機能　　ディレクトリを作成します。

形式　　# include <dos.h>

int mkdir(const char * *path*);

プロトタイプ　dir.h

解説　　**mkdir** は，*path* に指定されたパス名から，ディレクトリを新たに作成します。

戻り値　　**mkdir** は，新しいディレクトリが作成できた場合0を返します。

戻り値-1はエラーを意味し，***errno*** に次のいずれか値がセットされます。

EACCESS	アクセスは否定された
ENOENT	そのようなファイルまたはディレクトリはない

関連項目　　**chdir, getcurdir, getcwd, rmdir**

MK_FP

機能 far ポインタを作ります。

形式 # include <dos.h>

void far * MK_FP(unsigned *seg*, unsigned *ofs*) ;

プロトタイプ dos.h

解説 **MK_FP** は，セグメント（*seg*）とオフセット（*ofs*）から far ポインタを作り出すマクロです。

戻り値 **MK_FP** は far ポインタを返します。

関連項目 **FP_OFF, FP_SEG, movedata, segread**

例 **FP_OFF** を参照してください

mktemp

機能　ユニークなファイル名を作ります。

形式　char ＊ mktemp(char ＊ *template*)；

プロトタイプ　dir.h

解説　**mktemp** は，*template* が指す文字列をユニークなファイル名で置き換えて，*template* を返します。
template は，'*X*'6個からなり，ヌルで終わる文字列でなければなりません。6個の *X* は，ユニークな英字と1個のピリオドに置き換えられます。2個の英字，ピリオド，3個の英字になるわけです。
新しいファイル名は，AA.AAA から始めて，ディスク上のファイルの名前を見て，同じ名前にならないように選ばれます。

戻り値　*template* がうまく得られた場合は，**mktemp** は文字列 *template* のアドレスを返します。そうでない場合は NULL を返します。

可搬性　**mktemp** は UNIX システムで使用できます。

modf

機能　浮動小数点数を整数部と小数部に分割します。

形式　# include <math.h>

double modf(double *x*, double * *ipart*)；

プロトタイプ　math.h

解説　**modf** は，double 型の *x* を2つの部分(整数部と小数部)に分割します。*ipart* が指す場所に整数部を格納し，小数部を関数値として返します。

戻り値　**modf** は，*x* の小数部を返します。

関連項目　**fmod, ldexp**

movedata

機能　バイトをコピーします。

形式　void movedata(unsigned *srcseg*, unsigned *srcoff*, unsigned *dstseg*, unsigned *dstoff*, size_t *n*);

プロトタイプ　mem.h,string.h

解説　**movedata** は，アドレス *srcseg*：*srcoff* から *n* バイトを，*dstseg*：*dstoff* にコピーします。

movedata は，メモリモデルに依存しないデータブロックの転送手段です。

戻り値　ありません。

関連項目　**FP_OFF**, **memcpy**, **MK_FP**, **movmem**, **segread**

例

```
#include <mem.h>

#define RED_PLANE 0xA800

char buf[80*400];

/* グラフィックVRAMの赤のプレーンを buffer にセーブする */
Void save_red_plane(char near *buffer)
{
    movedata(RED_PLANE, 0, _DS, (unsigned) buffer, 80*400);
}

main()
{
    save_red_plane(buf);
}
```

movmem

機能　メモリブロックを指定の長さだけコピーします。

形式　void moemem(void * *src*, void * *dest*, unsigned *length*)；

プロトタイプ　mem.h

解説　**movmem** は，*length* バイトのブロックを，*src* から *dest* へコピーします。2つのブロックがオーバーラップしている場合は，正しくコピーされるようにコピーの方向が選ばれます。

戻り値　ありません。

関連項目　**memcpy**, **memmove**, **movedata**

nosound

機能　PC-9801：ダミー関数。
IBM PC：スピーカを止めます。

形式　void nosoud(void)；

プロトタイプ　dos.h

解説　PC-9801では，**nosound** は何の動作もしないダミー関数として定義されています。
IBM PC では，**nosound** は，**sound** の呼び出しによって鳴っているスピーカを止めます。

戻り値　ありません。

関連項目　**delay**, **sound**(IBM PC)

_open

機能　ファイルを読み出しまたは書き込み用にオープンします。

形式　# include <fcntl.h>

int _open(const char * *filename*, int *oflags*);

プロトタイプ　io.h

解説　**_open** は, *filename* に指定されたファイルをオープンし, *oflags* の値にしたがって読み出し，書き込み，あるいは両用として準備を行ないます。ファイルは，グローバル変数 ***_fmode*** に指定されているモードでオープンされます。

oflags の値は, DOS 2.*x* の下では, O_RDONLY, O_WRONLY, O_RDWR のいずれかに限定されます。DOS 3.*x* では，さらに以下の値を指定することができます。

O_NOINHERIT	そのファイルを子プロセスに引き継がないときに指定する。
O_DENYALL	現在のハンドルだけがそのファイルにアクセスできる。
O_DENYWRITE	そのファイルに対する読み出し用オープンのみを許す。
O_DENYREAD	そのファイルに対する書き込み用オープンのみを許す。
O_DENYNONE	そのファイルに対する他のシェアドオープンを許す。

これらの O_... シンボリック定数は fcntl.h の中で定義されています。

DOS 3.*x* の下では，1つの **_open** に対して，O_DENY*xxx* 値のうち，1つだけが指定できます。これらのファイルシェアリング属性は，ファイルのロック**機能**に加えて用意されています。同時にオープンできるファイル数の上限は，システムのコンフィギュレーションパラメータ(CONFIG.SYS ファイル中の FILES の値）になっています。

戻り値　成功すると，**_open** は負でない整数（ファイルハンドル）を返します。ファイル中の現在位置を示すファイルポインタは，ファイルの先頭にセットされます。エラーの場合は-1を返し，***errno*** に次のいずれかの値をセットします。

ENOENT	パス名またはファイル名が見つからなかった
EMFILE	オープンされているファイルが多すぎる
EACCESS	アクセスが否定されている
EINVACC	アクセスコードが正しくない

可搬性　**_open** は MS-DOS に特有の関数です。

関連項目　**open, _read, sopen**

open

機能　ファイルを読み出しまたは書き込み用にオープンします。

形式

<fcntl.h>の行:

```
#include <fcntl.h>
#include <sys¥stat.h>
int open(const char *path, int access [, unsigned mode]);
```

プロトタイプ　io.h

解説

open は，*path* に指定されたファイルをオープンし，*access* の値にしたがって，読み出し，書き込み，あるいは両用として準備を行ないます。

access に指定する値は，次に示す2つのリストのフラグをビットごとに OR することによって作られます。リスト1からは1つのフラグしか選べません。リスト2からは，複数のフラグを組み合わせることができます。

リスト1：読み出し/書き込みフラグ

O_RDONLY	読み出し専用としてオープン
O_WRONLY	書き込み専用としてオープン
O_RDWR	読み出しおよび書き込み用としてオープン

リスト2：他のアクセスフラグ

O_NDELAY	使用されません。UNIX とのコンパチビリティを図るためのものです。
O_APPEND	これがセットされていると，ファイルポインタはまずファイルの終わりに位置づけられます。
O_CREAT	指定のファイルがすでに存在していれば，このフラグは意味を持ちません。存在していなければ，ファイルは新規作成され，*mode* のビットが **chmod** の場合と同様にして，属性ビットにセットされます。
O_TRUNC	ファイルが存在している場合は，その長さは0に削られ

	ます。ファイル属性はそのままで変化しません。
O_EXCL	使用されません。UNIX とのコンパチビリティを図るためのものです。
O_BINARY	このフラグは，明示的にバイナリモードでファイルをオープンすることを示すのに使用されます。
O_TEXT	このフラグは，明示的にテキストモードでファイルをオープンすることを示すのに使用されます。

これらのシンボリック定数は，fcntl.h の中で定義されています。
O_BINARY も O_TEXT も与えられていない場合は，ファイルは，グローバル変数 ***_fmode*** にセットされている変換モードでオープンされます。
特定のモードでファイルを新たに作成するには，そのモード値を ***_fmode*** に代入しておくか，O_CREAT および O_TRUNC と要求するモードの OR をとったものを **open** の引数 *access* として与える必要があります。たとえば次の呼び出しは，

```
open("xmp", O_CREAT|O_TRUNC|O_BINARY, S_IREAD)
```

ファイル XMP を，バイナリモードで読み出し専用で作成します。XMP がすでに存在している場合は，ファイルサイズを0バイトにします。
O_CREAT フラグが *mode* を作るのに使われると，オプションの引数 *permiss* を次に示すシンボリック定数から作ることになります。

mode **の値**	**アクセス許可**
S_IWRITE	書き込み可
S_IREAD	読み出し可
S_IREAD ¦ S_IWRITE	読み書き可

戻り値

成功すると，**open** は負でない整数（ファイルハンドル）を返します。ファイルの現在位置を示すファイルポインタは，ファイルの先頭に位置づけられます。エラーの場合は-1を返し，***errno*** に次のいずれかの値をセットします。

ENOENT	パス名またはファイル名が見つからなかった
EMFILE	オープンされているファイルが多すぎる
EACCESS	アクセスが拒否された
EINVACC	アクセスコードが正しくない

可搬性

open は UNIX システムで使用可能です。UNIX バージョン7では，O_型のニーモニックは定義されていません。UNIX システム III は，O_BINARY を除くすべての O_型ニーモニックを使用しています。

関連項目

chmod, **chsize**, **close**, **creat**, **creatnew**, **creattemp**, **dup**, **dup2**, **fdopen**, **filelength**, **fopen**, **freopen**, **getftime**, **lock**, **_open**, **read**, **sopen**, **_write**

outport

機能　　ハードウェアポートに1ワード出力します。

形式　　void outport(int *portid*,int *value*)；

プロトタイプ　　dos.h

解説　　**outport** は，*portid* に指定された出力ポートへ，*value* に与えられたワードを出力します。

戻り値　　ありません。

可搬性　　**outport** は80x86ファミリィに特有の関数です。

関連項目　　**inport**, **inportb**, **outportb**

outportb

機能　ハードウェアポートに1バイト出力します。

形式

```
#include <dos.h>
void outportb(int portid, unsigned char value);
```

プロトタイプ　dos.h

解説　**outportb** は, *portid* に指定された出力ポートへ, *value* に指定されたバイトを出力するマクロです。

outportb は, dos.h がインクルードされている場合は, インラインコードに展開されるマクロとして扱われます。

dos.h をインクルードしていない場合や, インクルードしていてもマクロ **outportb** を# undef している場合は, 関数として扱われます。

戻り値　ありません。

可搬性　**outportb** は80x86ファミリィに特有の関数です。

関連項目　**inport**, **inportb**, **outport**

parsfnm

機能 ファイル名を解析します。

形式 #include <dos.h>

char * parsfnm(const char * *cmdline*, struct fcb * *fcb*, int *opt*);

プロトタイプ dos.h

解説 **parsfnm**は，*cmdline*が指す文字列を解析してファイル名を取りだします。*cmdline*は，通常はコマンドラインです。ファイル名はFCBの中に，ドライブ，ファイル名，および拡張子として配置されます。*fcb*がFCBを指しています。

引数*opt*は，DOSの解析システムコール0x29でALに与える値です。システムコール0x29における解析処理の詳細については，MS-DOSプログラマーズリファレンスマニュアルを参照してください。

戻り値 ファイル名の解析に成功した場合，**parsfnm**は，ファイル名の終わりの次のバイトを指すポインタを返します。エラーが起こった場合は0を返します。

可搬性 **parsfnm**はMS-DOSに特有の関数です。

peek

機能　指定されたメモリ位置のワードを返します。

形式　int peek(unsigned *segment*, unsigned *offset*);

プロトタイプ　dos.h

解説　**peek** は，*segment*：*offset* のメモリ位置にあるワードを返します。
peek は，dos.h をインクルードしている場合は，インラインコードに展開されるマクロとして扱われ，dos.h をインクルードしていない(あるいはインクルードしていても，#undef **peek** としている）場合は，関数として扱われます。

戻り値　**peek** は，*segment*：*offset* のメモリ位置にストアされているワードを返します。

可搬性　**peek** は80x86アーキテクチャに特有の関数です。

関連項目　**harderr**, **peekb**, **poke**

peekb

機能　指定されたメモリ位置のバイトを返します。

形式　#include <dos.h>
char peekb(unsigned *segment*, unsigned *offset*)；

プロトタイプ　dos.h

解説　**peekb** は，*segment*：*offset* のメモリ位置にあるバイトを返します。
peekb は，dos.h をインクルードしている場合は，インラインコードに展開されるマクロとして扱われ，dos.h をインクルードしていない(あるいはインクルードしていても，#undef **peekb** としている)場合は，関数として扱われます。

戻り値　**peekb** は，*segment*：*offset* のメモリ位置にストアされているバイトを返します。

可搬性　**peekb** は80x86アーキテクチャに特有の関数です。

関連項目　**peek,pokeb**

perror

機能　システムエラーメッセージをプリントします。

形式　void perror(const char ＊ *s*)；

プロトタイプ　stdio.h

解説　**perror** は，現在のプログラム内のシステムコールで起きた一番新しいエラーを示すメッセージを，ストリーム *stderr*(通常はコンソール)に出力します。
まず引数 *s* をプリントし，その後にコロン，***errno*** の現在の値に対応するメッセージを続け，最後に改行を出力します。慣例にしたがえば，引数 *s* にはプログラムの名前を指定することになります。
エラーメッセージ文字列の配列は，***sys_errlist*** を通じてアクセスされます。***errno*** を，エラー番号に対応する文字列を見つけ出すための添字として使うことができます。文字列には改行文字は含まれていません。
sys_nerr は，配列内のメッセージの数を示しています。
errno，***sys_errlist***，***sys_nerr*** の詳細については，第1章の「**グローバル変数**」を参照してください。

戻り値　ありません。

可搬性　**perror** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**clearerr**, **eof**, **_strerror**, **strerror**

poke

機能 指定のメモリ位置に整数値を格納します。

形式 void poke(unsigned *segment*, unsigned *offset*, int *value*)；

プロトタイプ dos.h

解説 **poke** は，*segment*：*offset* のメモリ位置に，整数値 *value* を格納します。**poke** は，dos.h をインクルードしている場合には，インラインコードに展開されるマクロとして扱われます。dos.h がインクルードされていない(あるいはインクルードされていても，#undef **poke** としている）場合は，マクロではなく関数として扱われます。

戻り値 ありません。

可搬性 **poke** は80x86ファミリィに特有の関数です。

関連項目 **harderr**, **peek**, **pokeb**

pokeb

機能 指定のメモリ位置にバイト値を格納します。

形式 # include <dos.h>
void pokeb(unsigned *segment*, unsigned *offset*, char *value*);

プロトタイプ dos.h

解説 **pokeb** は，*segment*：*offset* のメモリ位置に，バイト値 *value* を格納します。
pokeb は，dos.h をインクルードしている場合には，インラインコードに展開されるマクロとして扱われます。dos.h がインクルードされていない（あるいはインクルードされていても，# undef **pokeb** としている）場合は，マクロではなく関数として扱われます。

戻り値 ありません。

可搬性 **pokeb** は80x86ファミリィに特有の関数です。

関連項目 **peekb**, **poke**

poly

機能　多項式の計算

形式

```
#include <math.h>
double poly(double x, int degree, double coeffs[]);
```

プロトタイプ　math.h

解説　**poly** は，係数 *coeffs*[0]，*coeffs*[1]，...，*coeffs*[*degree*]，次数 *degree* の多項式の x における値を計算します。たとえば，*degree*＝4の場合，生成される多項式は次のようになります。

$$coeffs[4]x^4 + coeffs[3]x^3 + coeffs[2]x^2 + coeffs[1]x + coeffs[0]$$

戻り値　**poly** は，x における多項式の値を返します。

可搬性　**poly** は UNIX システムで使用できます。

pow

機能 x の y 乗を計算します。

形式

```
# include <math.h>
double pow(double x, double y);
```

プロトタイプ math.h

解説 **pow** は，x^yを計算します。

戻り値 **pow** 関数は，成功した場合，計算結果 x^yを返します。

引数の値によっては，オーバーフローが起きたり，計算できないことがあります。計算結果がオーバーフローになる場合，**pow** は HUGE_VAL を返します。大きさが非常に大きい結果の場合は ***errno*** に次の値をセットします。

ERANGE　値域エラー

引数 x が0以下で，かつ y が整数でなかった場合には，***errno*** は次の値にセットされます。

EDOM　定義域エラー

引数 x と y がともに0であった場合には，**pow** は1を返します。

pow のエラー処理は，**matherr** 関数を使って変更することができます。

可搬性 **pow** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **exp**, **pow10**, **sqrt**

pow10

機能 10の p 乗を計算します。

形式
```
#include &lt;math.h&gt;
double pow10(int p);
```

プロトタイプ math.h

解説 **pow10** は，10^pを計算します。

戻り値 **pow10** 関数は，成功した場合は計算結果10^pを返します。
結果は内部では long double で計算されます。このため有効な引数であっても，アンダーフローまたはオーバーフローが起きることがあります。

可搬性 **pow10** は UNIX システムで使用できます。

関連項目 **exp**, **pow**

printf

機能　書式つき出力を *stdout* に書き出します。

形式　int printf(const char * *format* [, *argument*, ...]);

プロトタイプ　stdio.h

解説　**printf** は，*format* によって指される書式文字列中の書式指定を，*format* の後に続く各引数に適用し，書式化されたデータを *stdout* に出力します。書式指定は，後に続く引数と同じ数だけなければなりません。

書式文字列：

書式文字列は **printf** 関数に含まれていますが，これは関数がどのように引数を変換，書式化，プリントするかを指定するものです。書式文字列に対して充分な数の引数が必要です。引数がたりない場合は，結果は予測できないものになります。引数が多すぎる場合は，単に無視されるだけです。

書式文字列には，2種類のオブジェクトが含まれています。通常の文字と変換指定です。

■通常の文字は単に出力ストリームに1文字ずつ送られます。

■変換指定は，引数リストから引数を取り出し，書式化を行ないます。

変換指定：

printf の書式文字列は次のような形式をもっています。

```
% [フラグ] [印字幅] [.精度] [F|N|h|l|L] 型
```

各変換指定は%で始まります。%に続いて，以下に示すものが，この順で現われます。

■オプションのフラグ文字の並び
■オプションの印字幅指定子
■オプションの精度指定子
■オプションの入力サイズ修飾子
■変換指定文字

書式文字列のオプションの要素：

書式文字列におけるオプションの文字，指定子，修飾子によって行われる出力の書式化の概要を示します。

文字/指定子	何を制御するのか，または何を指定するのか
フラグ	左詰めか右詰めか，数値の符号，小数点，後に続くゼロ，8進/16進のプレフィクス
印字幅	印字する最小文字数，余白をゼロ/空白どちらで埋めるか
精度	印字する最大文字数；整数の場合は印字する最小桁数
サイズ	引数のデフォルトサイズを変更する （**N**=**near** ポインタ，**F**=**far** ポインタ，**h**=**short int**，**l**=**long**，**L**=**long double**）

printf の変換指定文字：

次の表には，**printf** の変換指定文字，許される引数の型，の書式が示されています。

表中の情報は，書式指定には，フラグ文字，印字幅指定，精度指定，入力サイズ指定が含まれていないと仮定した場合に基づいています。オプションの文字および指定子がどのような影響を与えるかについては，この後の表を参照してください。

型指定文字	入力引数	出力の書式
	数値	
d	整数	符号つき10進整数
i	整数	符号つき10進整数
o	整数	符号なし8進整数
u	整数	符号なし10進整数
x	整数	符号なし16進整数（*a,b,c,d,e,f* を使用）
X	整数	符号なし16進整数（*A,B,C,D,E,F* を使用）
f	浮動小数点数	符号つきの[-]*dddd.dddd* 形式の値
e	浮動小数点数	符号つきの[-]*d.dddde*[+/-]*ddd* 形式の値
g	浮動小数点数	符号つきの *e* または *f* 形式の値，どちらになるかは与えられた値と精度によって決まる。後に続くゼロおよび小数点は必要な場合にだけ印字される。
E	浮動小数点	*e* と同じ，指数部を示す記号は *E* になる。
G	浮動小数点	*g* と同じ，*e* 形式が使用されるときは指数部を示す記号は *E* になる。

	文字	
c	文字	一個の文字。
s	文字列ポインタ	ヌル文字に出会うか，精度に達するまで文字をプリントする。
%	なし	文字%がプリントされる。
	ポインタ	
n	整数ポインタ	これまで書かれた文字の数を（入力引数によって指された場所に）格納する。
p	ポインタ	入力引数をポインタとして印字する。far ポインタは XXXX：YYYY，near ポインタは YYYY（オフセットだけ）と印字される。

慣例：

これらの指定子のいくつかに関しては次に示すような慣例があります。

文字	慣例
e,E	引数は[-]*d.ddd...e*[+/-]*ddd* 形式に変換される。 ・小数点の前には数字が1桁だけ置かれる ・小数点の後には「精度」桁の数字が置かれる ・指数部は常に3桁
f	引数は10進で[-]*ddd.ddd...*形式に変換され，小数点の後の桁数は精度と等しくなる（精度が0でないとき）。
g,G	引数は *e,E,f* のいずれかの形式で印字される。精度が有効数字の桁数を決める。後続の0は省略され，小数点は必要なときにのみつけられる。 変換指定文定が *g* の場合は，*e* または *f* 形式で印字される。*G* の場合は *E* 形式で印字される。変換の結果得られる指数部が次の場合には *e* 形式が使われる。 (a) 精度より大きい (b) −4より小さい
x,X	*x* 変換では，文字 *a,b,c,d,e,f* が出力中に現われ，*X* 変換では *A,B,C,D,E,F* が現われる。

注意：浮動小数点数の無限値は，+INF あるいは−INF としてプリントされます。また，IEEE の非数（Not−a−Number）は，+NAN あるいは−NAN としてプリントされます。

フラグ文字：

フラグとして使用される文字は，-，+，#，空白です。これらは組み合わせて指定できます。

フラグ	何を指定するか
−	結果を左詰めで印字する。つまり右側に空白を埋め込む。この指定がないと結果は右詰めで印字される。つまり，左側にゼロまたは空白を埋め込む。
+	符号つき変換の結果は，常に＋か－の符号がつけられてプリントされる。
空白	値が負でない場合は，正符号をつけずに空白で始まる。負の場合は負符号で始まる。
#	引数が交替形式を用いて変換されることを指定します。次の表を参照。

注意：＋と空白が両方指定されている場合は＋が優先します。

交替形式：

#が変換文字とともに使用された場合は，変換される引数に対して次のような効果をもちます。

変換文字	#がどんな影響を与えるか
c,s,d,i,u	影響なし
o	0でない引数の場合，その前に0がつけられる。
x,X	0x（または0X）が引数の前につく。
e,E,f	出力には，その後に数字がなくても常に小数点が含まれる。通常は，後に数字があるときにのみ小数点が置かれる。
g,G	*e,E* の場合と同じ。ただし最後のゼロは省略される。

印字幅指定子：

印字幅指定子は，出力のための最小のフィールド幅をセットします。

印字幅の指定する方法は2通りあります。数字列を使って直接的に指定するものと，アスタリスク（*）を使って間接的に指定するものがあります。印字幅指定子としてアスタリスクを使う場合，実際に出力される引数の前に，印字幅指定子を含む引数（int 型でなければならない）が指定されていなければなりません。

印字幅が小さくても，フィールドが削られることはありません。変換の結果がフィールド幅より広い場合は，フィールドは単にひろげられ，変換された結果が入ります。

印字幅指定子	出力幅はどのようになるか
n	少なくとも *n* 文字印字される。出力値が *n* 文字より少ないときは空白で埋められる（-フラグがある場合は右側，ない場合は左側）。
0**n**	少なくとも n 文字印字される。出力値が n 文字より少ないときはゼロで埋められる（-フラグがある場合は右側，ない場合は左側）。
*	引数リストは印字幅指定子を含む。印字幅指定子は書式化される実引数の前に置かなくてはならない。

精度指定子：

精度指定子は常にピリオド（.）で始まり，印字幅指定子とは区別されます。印字幅の場合と同様，精度も，数字列を使って直接，あるいはアスタリスク（*）を使って間接に指定することができます。精度指定子にアスタリスクを使う場合，実際に出力される引数の前に，精度指定子を含む引数（int 型でなければならない）を指定しなければなりません。

アスタリスクを，印字幅指定子と精度指定子の両方に指定した場合は，実際に変換される引数の前に2つの引数が置かれることになります。

精度指定子	出力精度はどのようになるか
指定なし	デフォルトの精度 d,i,o,u,x,X 形式　1 e,E,f 形式　6 g,G 形式　有効数字すべて s 形式　ヌル文字までプリントする c 形式　影響を与えない
.0	d,i,o,u,x 形式に対しては精度はデフォルト値にセットされる。e,E,f 形式に対しては小数点は出力されない。
.n	*n* 文字あるいは *n* 桁が印字される。出力値が *n* 文字以上の場合は，出力は（変換指定文字によって）削られたり丸められたりすることがある。
*	引数リストは精度指定子を含んでいる。精度指定子は書式化される実引数の前に置かなくてはならない。

注意：明示的に精度0が指定され，そのフィールドの書式指定が整数形式（d,i,o,u,x）で，なおかつ印字される値が0の場合には，そのフィールドには数字はまったく出力されません（つまりそのフィールドは空白になります）。

変換文字	精度指定子はどのような影響を与えるか
d,i,o,u,x,X	*.n* は，最低 n 桁が印字されることを指定する。入力引数が *n* 桁より少ない場合は，出力値は左側がゼロで埋められる。入力引数が *n* 桁より大きくても出力値は削られない。
e,E,f	*.n* は，小数点の後に *n* 文字印字されることを指定する。最後の桁は丸められる。
g,G	*.n* は，最大 *n* 桁の有効数字が印字されることを指定する。
c	*.n* は，出力に関して何の影響も与えない。
s	*.n* は，*n* 文字以上の文字が印字されないことを指定する。

入力サイズ修飾子：

入力サイズ修飾子の文字（F,N,h,l,L）は，次に示す入力引数のサイズを与えます。

F = far ポインタ　　l = long
N = near ポインタ　L = long double
h = short int

入力サイズ修飾子（F,N,h,l,L）は，対応する入力引数のデータ型を... **printf** 関数がどのように解釈するかに影響を与えます。F と N は，ポインタ（%p,%s,%n）の引数に対してのみ使用することができます。h, l および L は，数値（整数と浮動小数点数）の入力引数に対してのみ適用できます。

F と N はともに，入力引数を再解釈するためのものです。通常は% p,% s,% n 変換に対応する引数は，使用中のメモリモデルのデフォルトサイズのポインタです。F は「入力引数を far ポインタとして解釈する」，N は「入力引数を near ポインタとして解釈する」という意味です。

h, l および L はともに，入力数値データ引数のデフォルトのサイズを変更するためのものです。l と L は整数（d,i,o,u,x,X）と浮動小数点数（e, E,f,g,G）の各形式に対して適用できるのに対し，h は整数形式にしか適用できません。h, l および L は，文字の c,s 形式や，ポインタの p,n 形式には影響を与えません。

入力サイズ修飾子	引数がどう解釈されるか
F	引数は far ポインタとして解釈される。
N	引数は near ポインタとして解釈される。N はヒュージモデルでは，どんな変換とも組み合わせて使うことはできません。
h	d,i,o,u,x,X では，引数は short int として解釈される。
l	d,i,o,u,x,X では，引数は long int と解釈される。 e,E,f,g,G では，引数は double と解釈される。
L	e,E,f,g,G では，引数は long double と解釈される。

戻り値　printf は出力したバイト数を返します。エラーの場合は EOF を返します。

可搬性　printf は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**cprintf, ecvt, fprintf, fread, fscanf, putc, puts, putw, scanf, sprintf, vprintf, vsprintf**

例

```
#include <stdio.h>
#define I 555
#define R 5.5

main()  {
   int  i, j, k, l;
   char buf[7];
   char *prefix = buf;
   char tp[20];

   printf("prefix    6d     6o       8x         10.2e        10.2f¥n");
   strcpy(prefix,"%");
   for (i = 0; i < 2; i++)  {
      for (j = 0; j < 2; j++)
         for (k = 0; k < 2; k++)
            for (l = 0; l < 2; l++)  {
               if (i == 0)  strcat(prefix, "-");
               if (j == 0)  strcat(prefix, "+");
               if (k == 0)  strcat(prefix, "#");
               if (l == 0)  strcat(prefix, "0");
               printf("%5s |", prefix);
               strcpy(tp, prefix);
               strcat(tp, "6d |");
               printf(tp, I);
               strcpy(tp, "");
               strcpy(tp, prefix);
               strcat(tp, "6o |");
               printf(tp, I);
               strcpy(tp, "");
               strcpy(tp, prefix);
               strcat(tp, "8x |");
               printf(tp, I);
               strcpy(tp, "");
               strcpy(tp, prefix);
               strcat(tp, "10.2e |");
               printf(tp, R);
               strcpy(tp, prefix);
               strcat(tp, "10.2f |");
               printf(tp, R);
               printf("  ¥n");
               strcpy(prefix, "%");
            }
   }
}
```

プログラム出力

```
prefix    6d      6o        8x        10.2e       10.2f
%-+#0 |+555   |01053  |0x22b    |+5.50e+00  |+5.50      |
 %-+# |+555   |01053  |0x22b    |+5.50e+00  |+5.50      |
 %-+0 |+555   |1053   |22b      |+5.50e+00  |+5.50      |
  %-+ |+555   |1053   |22b      |+5.50e+00  |+5.50      |
 %-#0 |555    |01053  |0x22b    |5.50e+00   |5.50       |
  %-# |555    |01053  |0x22b    |5.50e+00   |5.50       |
  %-0 |555    |1053   |22b      |5.50e+00   |5.50       |
   %- |555    |1053   |22b      |5.50e+00   |5.50       |
 %+#0 |+00555 |001053 |0x00022b |+05.50e+00 |+000005.50 |
  %+# |  +555 | 01053 |    0x22b | +5.50e+00 |      +5.50 |
  %+0 |+00555 |001053 |0000022b |+05.50e+00 |+000005.50 |
   %+ |  +555 |  1053 |      22b | +5.50e+00 |      +5.50 |
  %#0 |000555 |001053 |0x00022b |005.50e+00 |0000005.50 |
   %# |   555 | 01053 |    0x22b |  5.50e+00 |       5.50 |
   %0 |000555 |001053 |0000022b |005.50e+00 |0000005.50 |
    % |   555 |  1053 |      22b |  5.50e+00 |       5.50 |
```

putc

機能 ストリームに1文字を出力します。

形式 # include<stdio.h>

int putc(int *c*, FILE * *stream*);

プロトタイプ stdio.h

解説 **putc** は，文字 *c* を *stream* に指定されたストリームへ出力するマクロです。

戻り値 成功した場合, **putc** は出力した文字 *c* を返します。エラーの場合は EOF を返します。

可搬性 **putc** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目 **fprintf**, **fputc**, **fputch**, **getc**, **getchar**, **printf**, **putch**, **putchar**

putch

機能　画面に1文字を出力します。

形式　int putch(int *c*);

プロトタイプ　conio.h

解説　**putch** は，文字 *c* をカレントテキストウインドウに出力します。**putch** は，コンソールへの直接出力を行なうテキストモード関数です。**putch** は，改行文字（¥n）を CR/LF ペアに変換しません。

戻り値　成功した場合，**putch** は印字した文字 *c* を返します。エラーの場合には EOF を返します。

可搬性　**putch** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。

関連項目　**cprintf, cputs, getch, getche, putc, putchar**

putchar

機能 *stdout* へ文字を出力します。

形式
```
#include <stdio.h>
int putchar(int c);
```

プロトタイプ stdio.h

解説 **putchar**(*c*)は，**putc**(*c*, *stdout*)と定義されたマクロです。

戻り値 成功した場合，**putchar** は文字 *c* を返します。エラーの場合には EOF を返します。

可搬性 **putchar** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目 **fputchar**, **getc**, **getchar**, **putc**, **putch**, **puts**

putenv

機能 現在の環境変数に文字列を追加します。

形式 int putenv(const char * *name*);

プロトタイプ stdlib.h

解説 **putenv** は，文字列 *name* を引数にとり，それを現在のプロセスの環境に追加します。たとえば次のように呼び出します。

```
putenv("PATH=A:¥FOO");
```

putenv は，すでにある *name* を変更または削除するためにも使用できます。削除するときは，その変数の値を空にします（たとえば"MYVAR="のように指定します)。

putenv が変更するのは現在のプログラムの環境だけです。プログラムが終了すれば元の環境が回復されます。

戻り値 **putenv** は，成功した場合は0を返し，エラーの場合は-1を返します。

可搬性 **putenv** は UNIX システムで使用できます。

関連項目 **getenv**

puts

機能　*stdout* に文字列を出力します。

形式　int puts(const char ＊ *s*)；

プロトタイプ　stdio.h

解説　**puts** は，ヌル文字で終わる文字列 *s* を，標準出力ストリーム *stdout* に出力し，最後に改行文字をつけます。

戻り値　**puts** は，成功した場合は負でない値を返し，エラーの場合は EOF を返します。

可搬性　**puts** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**cputs**, **fputs**, **gets**, **printf**, **putchar**

putw

機能　ストリームに整数を出力します。

形式　# include <stdio.h>

int putw(int *w*, FILE * *stream*);

プロトタイプ　stdio.h

解説　**putw** は，整数 *w* を出力ストリーム *stream* に出力します。**putw** は，ファイルの中での特別なアラインメントは想定しておらず，また行ないません。

戻り値　**putw** は，成功した場合は整数 *w* を返します。エラーの場合はは EOF を返します。

EOF は **putw** が正常な値として返す整数なので，**putw** のエラーを検出するには **ferror** を使用するべきです。

可搬性　**putw** は UNIX システムで使用できます。

関連項目　**getw**, **printf**

qsort

機能 クイックソートアルゴリズムを用いてソートを行ないます。

形式

```
void qsort(void * base, size_t nelem, size_t width,
           int (* fcmp)(const void *, const void *));
```

プロトタイプ stdlib.h

解説

qsort は，クイックソートアルゴリズムの1つ“3つのメディアン”を実現したものです。**qsort** は，*fcmp* が指すユーザ定義の比較関数を繰り返し呼び出して，テーブル中の項目をソートします。

- *base* は，ソートされるべきテーブルのベース(つまり0番目の要素)を指しています。
- *nelem* はテーブル中の項目の数です。
- *width* はテーブル中の各項目の大きさ（単位：バイト）です。

比較関数＊ *fcmp* は，2つの引数 *elem1* と *elem2* をとります。2つの引数はテーブル中の項目へのポインタです。比較関数はこの2つの項目(＊ *elem1*,＊ *elem2*）を比較し，その比較結果を整数で返します。

比較結果	*fcmp* の戻り値
＊ *elem1* 〈 ＊ *elem2*	負の整数
＊ *elem1* == ＊ *elem2*	0
＊ *elem1* 〉 ＊ *elem2*	正の整数

不等号〈は，左側の項目が，最終的にソートされた並びにおいて，右側の項目より前に来るべきであることを意味します。同様に，〉は左側の項目が，最終的にソートされた並びにおいて，右側の項目より後に来るべきであることを意味します。

戻り値　　ありません。

可搬性　　**qsort** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**bsearch**, **lsearch**

例

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char list[5][4] = { "cat", "car", "cab", "cap", "can" };

main()
{
   int v;

   qsort(&list, 5, sizeof(list[10]), strcmp);
   for (x = 0; x < 5; x++)
      printf("%s¥n", list[x]);
}
```

プログラム出力

```
cab
can
cap
car
cat
```

raise

機能　実行中のプログラムにソフトウェアシグナルを送ります。

形式

```
#include <signal.h>
int raise(int sig);
```

プロトタイプ　signal.h

解説　**raise** は，*sig* 型のシグナルをプログラムに送ります。*sig* によって指定されるシグナル型に対するシグナルハンドラがインストールされている場合は，そのハンドラが実行されます。ハンドラがインストールされていない場合は，そのシグナル型に対するデフォルトの処理が行なわれます。
シグナル型は，signal.h の中で次のように定義されています。

シグナル	意味
SIGABRT	異常終了（*）
SIGFPE	正しくない浮動小数点演算
SIGILL	不当な命令（#）
SIGINT	コントロールブレーク割り込み
SIGSEGV	主記憶に対する正しくないアクセス（#）
SIGTERM	プログラム終了の要求（*）

アスタリスク * がついているシグナル型は，通常の操作では DOS や Turbo C によって生成されることはありません。それらは **raise** で生成されるものです。シャープ#がついているシグナルは，8088や8086プロセッサでは非同期的に生成することはできませんが，他のプロセッサでは生成されることがあります（詳しくは **signal** の解説を参照してください）。

戻り値　成功した場合は0，そうでない場合は0以外の値を返します。

可搬性　**raise** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**abort**, **signal**

例

```
#include <signal.h>

main()
{
   int a, b, c;

   a = 10;
   b = 0;
   if (b == 0)
      raise(SIGFPE);      /* ゼロ除算エラーを取り除く */
   c = a / b;
}
```

rand

機能　乱数発生ルーチンです。

形式　int rand(void)；

プロトタイプ　stdlib.h

解説　**rand** は，周期 2^{32} の乗法合同法を用いて乱数を発生し，呼び出されるたびに0～RAND_MAX の間の疑似乱数を返します。シンボリック定数 RAND MAX は stdlib.h の中で定義されており，値は2 15±1です。

戻り値　**rand** は，生成した疑似乱数値を返します。

可搬性　**rand** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**random, randomize, srand**

例

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

main()  /* 0～32767 の範囲の5つの乱数を表示する */
{
   int i;

   srand(time(NULL) % 37);    /* ランダムな値から始める */
   for (i=0; i<5; i++)
       printf("%d¥n", rand());
}
```

randbrd

機能　ランダムなブロック読み出しを行ないます。

形式

```
#include <dos.h>
int randbrd(struct fcb *fcb, int rcnt);
```

プロトタイプ　dos.h

解説

randbrd は，*fcb* が指すオープンされているファイルコントロールブロック（FCB）を使って，*rcnt* 個のレコードを読み込みます。レコードは，現在のディスク転送アドレス(DTA)のメモリに，FCB のランダムレコードフィールドで指定されたディスクレコードから読まれます。これは DOS システムコール0x27を呼び出すことによって行なわれます。

実際に読み出されたレコードの数は，FCB のランダムレコードフィールドを調べることによって得ることができます。ランダムレコードフィールドは，実際に読み出されたレコード数だけ増加します。

戻り値

randbrd の結果によって次に示す値が返されます。

0	すべてのレコードが読み込まれました。
1	EOF に達し，最後のレコードは完全に読み込まれました。
2	読み込むレコードが0xFFFF を越えます（可能な最大数のレコードが読まれました）。
3	EOF に達し，最後のレコードは不完全でした。

可搬性　**randbrd** は MS-DOS に特有な関数です。

関連項目　**getdta, randbwr, setdta**

randbwr

機能　ランダムなブロック書き込みを行ないます。

形式　# include<dos.h>
int randbwr(struct fcb * *fcb*, int *rcnt*) ;

プロトタイプ　dos.h

解説　**randbwr** は，*fcb* が指すオープンされているファイルコントロールブロック（FCB）を使って，*rcnt* 個のレコードを書き込みます。レコードは，現在のディスク転送アドレス(DTA)のメモリから，FCB のランダムレコードフィールドで指定されたディスクレコードへ書き込まれます。これは，DOS システムコール0x28を呼び出すことによって行なわれます。*rcnt* が0の場合，ファイルの長さは，ランダムレコードフィールドが示している値にセットされます。
実際に書き込まれたレコードの数は，FCB のランダムレコードフィールドを調べることによって得ることができます。ランダムレコードフフィールドは，実際に読み出された，あるいは書き込まれたレコード数だけ増加します。

戻り値　**randbwr** の結果によって次に示す値が返されます。

0	すべてのレコードが書き込まれました。
1	レコードを書き込むだけのディスク領域がありません(レコードはいっさい書き込まれません)。
2	書き込もうとするレコードが0xFFFF を越えます（可能な最大数のレコードが書き込まれました)。

可搬性　**randbwr** は MS-DOS に特有な関数です。

関連項目　**randbrd**

random

機能　乱数発生ルーチンです。

形式　# include <stdlib.h>
int random(int *num*)；

プロトタイプ　stdlib.h

解説　**random** は，0と(*num*-1)の間の乱数を返します。**random**(*num*)は，マクロとして(**rand**() % (*num*))と定義されています。*num* と返される乱数はどちらも整数です。

戻り値　**random** は，0と(*num*-1)の間の数値を返します。

可搬性　これに対応する関数が Turbo Pascal にもあります。

関連項目　**rand, randomize, srand**

例

```
#include <stdlib.h>
#include <time.h>

/* 0～99 の範囲の乱数を，ランダムな個数出力する */
main()
{
   int n;

   randomize();
   /* 1～20 の範囲のランダムな値をとる */
   n = random(20) + 1;
   while (n-- > 0)
      printf ("%d ", random (100));
   printf("¥n");
}
```

randomize

機能　乱数発生ルーチンを初期化します。

形式

```
#include <stdlib.h>
#include <time.h>
void randomize(void);
```

プロトタイプ　stdlib.h

解説　**randomize**は，ランダムな値によって乱数発生ルーチンを初期化します。**randomize**は，ヘッダファイルtime.hでプロトタイプ宣言されているtime関数を呼び出すマクロなので，この関数を使う場合は，time.hをインクルードするようにしてください。

戻り値　戻り値はありません。

可搬性　これに対応する関数がTurbo Pascalにもあります。

関連項目　**rand**, **random**, **srand**

_read

機能　ファイルから読み込みを行ないます。

形式　int　_read(int *handle*, void ＊ *buf*, unsigned *len*)；

プロトタイプ　io.h

解説　**_read** は，*handle* に結びつけられているファイルから *len* バイトを読み込んで，*buf* が指しているバッファに格納します。**_read** は，DOS システムコールを直接呼び出します。

テキストモードでオープンされたファイルに対しては，**_read** は復帰文字（0x0D）を削除しません。

handle は，**creat**，**open**，**dup**，**dup2** の呼び出しで得られたファイルハンドルです。

ディスクファイルでは，**_read** は現在のファイルポインタの位置から読み込みを開始します。読み込みが終了したときに，ファイルポインタを読み込んだバイト数だけインクリメントします。デバイスの場合は，バイトはデバイスから直接読まれます。

_read で読み込めるバイト数は最大65534バイトです。65535 (0xFFFF) は -1と同じであり，エラーを示すために使用されます。

戻り値　成功した場合，**_read** はバッファに格納されたバイト数を示す正の整数を返します。ファイルエンドを読み込んだ場合は0を返します。エラーの場合は-1を返し，*errno* に次のいずれかの値をセットします。

EACCES	アクセスが拒否された
EBADF	ファイル番号が正しくない

可搬性　**_read** は MS-DOS に特有の関数です。

関連項目　**_open**，**read**，**_write**

read

機能　ファイルから読み込みを行ないます。

形式　int read(int *handle*, void * *buf*, unsigned *len*);

プロトタイプ　io.h

解説

read は，*handle* と結びつけられているファイルから *len* バイトを読み込んで，*buf* が指しているバッファに格納します。

テキストモードでオープンされたファイルに対しては，**read** は復帰文字を削除し，ファイルエンドを読み込んだときには，それを知らせます。

handle は，**creat**，**open**，**dup**，**dup2** の呼び出しで得られたファイルハンドルです。

ディスクファイルでは，**read** は，現在のファイルポインタの位置から読み込みを開始します。読み込みが終了したときに，ファイルポインタを読み込んだバイト数だけインクリメントします。デバイスの場合は，バイトはデバイスから直接読まれます。

read で読み込めるバイト数は最大65534バイトです。65535 (0xFFFF) は-1と同じであり，エラーを示すために使用されます。

戻り値

成功した場合，**read** は，バッファに格納されたバイト数を示す正の整数を返します。ファイルがテキストモードでオープンされている場合は，**read** は復帰文字および CTRL-Z 文字はカウントしません。

ファイルエンドを読み込んだ場合は0を返します。エラーの場合は-1を返し，*errno* に次のいずれかの値をセットします。

EACCES	アクセスが拒否された
EBADF	ファイル番号が正しくない

可搬性　**read** は UNIX システムで使用できます。

関連項目　**open**，**_read**，**write**

realloc

機能　メモリの再割り当てを行ないます。

形式
```
#include <stdlib.h>
void * realloc(void * block, size_t size) ;
```

プロトタイプ　stdlib.h，alloc.h

解説　**realloc** は，すでに確保されているブロックを，*size* バイトに縮小または拡張します。引数 *block* は，以前の **malloc**, **calloc**, あるいは **realloc** の呼び出しで得られたメモリブロックを指すポインタです。*block* がヌルポインタの場合には，**realloc** は **malloc** とまったく同じように機能します。

realloc は，割り当て済みのブロックをのサイズを *size* に調整し，必要であればブロックの内容を新しい領域にコピーします。

戻り値　**realloc** は，再割り当てされたブロックのアドレスを返します。これは，以前のブロックのアドレスとは異なる場合もあります。ブロックを再割り当てできなかった場合，あるいは *size*==0の場合には NULL を返します。

可搬性　**realloc** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**calloc**，**farrealloc**，**free**, **malloc**

例　**malloc** を参照してください。

remove

機能 ファイルを削除します。

形式 # include <stdio.h>

int remove(const char * *filename*) ;

プロトタイプ stdio.h

解説 **remove** は，*filename* に指定されたファイルを削除します。**remove** は，単に **unlink** の呼び出しに展開されるマクロです。

戻り値 成功した場合は0を返します。エラーの場合は-1を返し，***errno*** に次のいずれかをセットします。

ENOENT	そのようなファイルあるいはディレクトリはない
EACCES	アクセスが拒否された

可搬性 **remove** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **unlink**

rename

機能　ファイル名を変更します。

形式　int rename(const char * *oldname*, const char * *newname*)；

プロトタイプ　stdio.h

解説　**rename** は，ファイルの名前を *oldname* から *newname* に変更します。*newname* にドライブ名の指定がある場合は，*oldname* のものと同じでなければなりません。
パス名の中のディレクトリは同じである必要はありません。したがって，**rename** によって，ファイルをあるディレクトリから別のディレクトリに移すこともできます。ワイルドカードは使えません。

戻り値　成功した場合，**rename** は0を返します。エラーの場合は-1を返し，***errno*** に次の値のいずれかをセットします。

ENOENT	そのようなファイルまたはディレクトリはない
EACCES	アクセスが拒否された
ENOTSAM	同じデバイスではない

可搬性　**rename** は ANSI C と互換性があります。

rewind

機能　ファイルポインタをファイルの先頭に移動します。

形式
```
#include <stdio.h>
void rewind(FILE * stream);
```

プロトタイプ　stdio.h

解説　**rewind**(*stream*)は，**fseek**(*stream*, 0L, SEEK_SET)とほぼ同じです。ただし，**rewind** はファイル終了標識とエラー標識の両方をクリアしますが，**fseek** はファイル終了標識のみをクリアします。

更新用にオープンされたファイルでは，**rewind** を呼び出した後は，入力・出力どちらでも行なうことができます。

戻り値　ありません。

可搬性　**rewind** はすべての UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**fopen**，**fseek**，**ftell**

例　**fseek** を参照してください。

rmdir

機能 ディレクトリを削除します。

形式 int rmdir(const char * *path*);

プロトタイプ dir.h

解説 **rmdir** は，*path* に指定されたディレクトリを削除します。*path* が示すディレクトリ名は，

■空でなければなりません。
■現在の作業ディレクトリであってはなりません。
■ルートディレクトリであってはなりません。

戻り値 **rmdir** は，ディレクトリを削除できた場合には0を返します。エラーの場合は-1を返し，***errno*** に次のいずれかをセットします。

EACCESS	アクセスが否定されている
ENOENT	パス名またはファイル名が見つからない

関連項目 **chdir**, **getcurdir**, **getcwd**, **mkdir**

_rotl

機能　符号なし整数値をビットごとに左へローテートします。

形式　unsigned _rotl(unsigned *value*, int *count*)；

プロトタイプ　stdlib.h

解説　**_rotl** は，与えられた値 *value* を *count* ビットだけ左へローテートします。ローテートされる値は unsigned です。

戻り値　**_rotl** は，左へ *count* ビットローテートした値を返します。

関連項目　**_lrotl**

例

```
#include <stdlib.h>

main()
{
   printf("rotate 0xABCD 4 bits left = %04X¥n",
          _rotl(0xABCD, 4));
   printf("rotate 0xABCD 4 bits right = %04X¥n",
          _rotr(0xABCD, 4));
   printf("rotate 0x55555555 1 bit left = %08lX¥n",
          _lrotl(0x55555555L, 1));
   printf("rotate 0xAAAAAAAA 1 bit right = %08lX¥n",
          _lrotr(0xAAAAAAAAL, 1));
}
```

プログラム出力

```
rotate 0xABCD 4 bits left  = BCDA
rotate 0xABCD 4 bits right = DABC
rotate 0x55555555 1 bit left  = AAAAAAAA
rotate 0xAAAAAAAA 1 bit right = 55555555
```

_rotr

機能　符号なし整数値をビットごとに右へローテートします。

形式　unsigned _rotr(unsigned *value*, int *count*);

プロトタイプ　stdlib.h

解説　**_rotr** は，与えられた値 *value* を *count* ビットだけ右へローテートします。ローテートされる値は unsigned です。

戻り値　**_rotr** は，右へ *count* ビットローテートした値を返します。

関連項目　**_lrotr**

例　**_rotl** を参照してください。

sbrk

機能　データセグメント領域の割り当てを変更します。

形式　void * sbrk(int *incr*);

プロトタイプ　alloc.h

解説　**sbrk** は，ブレーク値（データセグメントのすぐ後ろのアドレス）に *incr* を加え，これに従って割り当てスペースを変更します。*incr* は負の値でもかまわず，その場合には割り当てスペースが小さくなります。
sbrk は，システムで許される大きさを越えるスペースを確保しようとした場合にはエラーとなり，割り当てスペースの変更は行ないません。

戻り値　割り当てスペースの変更に成功した場合，**sbrk** は変更前のブレーク値を返します。失敗した場合には-1を返し，*errno* を次の値にセットします。

ENOMEM　メモリが不足した

可搬性　**sbrk** は UNIX で使用できます。

関連項目　**brk**

scanf

機能　ストリーム *stdin* をスキャンして書式つき入力を行ないます。

形式　int scanf(const char * *format* [, *address*,...]);

プロトタイプ　stdio.h

解説　**scanf** は，一連の入力フィールドをスキャンして，一度に1文字ずつストリーム *stdin* から文字を読み込みます。次に，引数 *format* によって指される書式文字列中の書式指定にしたがって，各フィールドを書式化します。最後に，*format* の後に続く各引数が示しているアドレスに，書式化した入力を格納していきます。書式文字列中の書式指定の個数は，その後に続くアドレスの数と同じでなければなりません。

書式文字列：

scanf および関連する関数（**cscanf**, **fscanf**, **sscanf**, **vscanf**, **vfscanf**, **vsscanf**）の書式文字列は，各関数が入力フィールドを，どのようにスキャンし，変換し，また格納するかを制御します。書式指定と同じ個数のアドレス引数が存在しなければなりません。アドレス引数がたりない場合は結果は予測できない，ひどいものになるはずです。多すぎる場合は単に無視されるだけです。

書式文字列には，次の3種類のオブジェクトが含まれています。**ホワイトスペース**，**非ホワイトスペース**，**書式指定**です。

■ホワイトスペースは，空白，タブ（¥t），復改（¥n）です。

...**scanf** 関数が書式文字列中のホワイトスペースに出会うと，入力におけるホワイトスペースを，次に非ホワイトスペースが現われるまで読みとばします。

■非ホワイトスペースは，％文字以外のすべての ASCII 文字です。

...**scanf** 関数が書式文字列の中で非ホワイトスペースに出会うと，それに一致する文字を読みます。

■書式指定は，...**scanf** 関数に，入力フィールドを読んで変換し，アドレ

ス引数で指定される場所へ格納することを指示します。

後に続くホワイトスペースは，書式文字列の中でマッチするものがない限り，（復改も含めて）読まれません。

書式指定：

...**scanf** の書式指定は次のような形式をとります。

% [*] [入力幅] [F¦N] [h¦l¦L] 型指定文字

書式指定は必らず%で始まります。%の後には，次に示すものが，この順で現われます。

■オプションの代入抑制文字（*）
■オプションの入力幅指定子
■オプションのポインタサイズ指定子（F または N）
■オプションの引数型修飾子（h, l または L）
■型指定文字

書式文字列中のオプションの要素：

...**scanf** の書式文字列の中の指定するオプションの要素の働きを次に示します。

文字/指定子	働き
*	次の入力フィールドを読みとばす。
入力幅	読むべき文字数の最大値。...scanf 関数がホワイトスペースあるいは変換不能文字に出会った場合は，これより少ない文字しか読まれないこともある。
サイズ	アドレス引数のデフォルトのサイズを変更する。 *N* = near ポインタ *F* = far ポインタ
引数の型	アドレス引数のデフォルトの型を変更する。 *h* = short int *l* = long int（型指定文字が整数変換を指示している場合） *l* = double（型指定文字が浮動小数点変換を指示している場合） *L* = long double（浮動小数点変換の場合にのみ有効）

...scanf 型指定文字：

次の表には，**...scanf** の型指定文字，期待される入力の型，入力がどのような書式で格納されるかが示されています。

この表に示した情報は，オプションの文字，指定子，修飾子（*，入力幅，サイズ）が書式指定に指定されていない場合のものです。オプションを追加したときにどうなるかについては，この後の表を参照してください。

型指定文字	期待される入力	引数の型
	数値	
d	10進整数	int へのポインタ（int * *arg*）
D	10進整数	long へのポインタ（ling * *arg*）
o	8進整数	int へのポインタ（int * *arg*）
O	8進整数	long へのポインタ（long * *arg*）
i	10/8/16進整数	int へのポインタ（int * *arg*）
I	10/8/16進整数	long へのポインタ（long * *arg*）
u	符号なし10進整数	unsigned int へのポインタ（unsigned int * *arg*）
U	符号なし10進整数	unsigned long へのポインタ（unsigned long * *arg*）
x	16進整数	int へのポインタ(int * *arg*)
X	16進整数	long へのポインタ(long * *arg*)
e	浮動小数点数	float へのポインタ（*float* * *arg*）
E	浮動小数点数	float へのポインタ（float * *arg*）
f	浮動小数点数	float へのポインタ（float * *arg*）
g	浮動小数点数	float へのポインタ（float * *arg*）
G	浮動小数点数	float へのポインタ（float * *arg*）

文字		
s	文字列	文字配列へのポインタ（char *arg*[]）
c	文字	文字へのポインタ（char * *arg*） フィールド幅 **w** が **c** についている場合（%5c のように）は，w 個の文字からなる配列を指すポインタ（char *arg* [**w**]）
%	%文字	変換は，行なわれず，単に%文字が格納される。

ポインタ		
n	（なし）	int 型へのポインタ（int * *arg*） %n までで，正しく読み込まれた文字数がこの int に格納される。
p	16進数 YYYY：ZZZZ 形式 または ZZZZ 形式	あるオブジェクトへのポインタ (far *または near *) %**p** 変換は，メモリモデルにおけるデフォルトのポインタサイズになる。

入力フィールド：

入力フィールドは，次に示すもののいずれかです。

■次のホワイトスペース（これは含まれない）までのすべての文字

■現在の書式指定では変換することができない文字までのすべての文字（たとえば，8進形式での8または9）

■ *n* がフィールド幅として指定されている場合は *n* 文字まで。

慣例：

書式指定のいくつかについては，次に示すような慣例があります。

%c 変換

この指定子は，ホワイトスペースを含めて次の文字を読み込みます。ホワイトスペースをスキップし，次の非ホワイトスペースを読み込むためには，%1s を使用してください。

%Wc 変換（W ＝ 入力幅指定子）

アドレス引数は文字配列を指すポインタです。配列は W 個の要素をもっています（char *arg*[**W**]）。

%s 変換

アドレス引数は文字配列を指すポインタです（char *arg*[]）。
配列の大きさは少なくとも（n+1）バイトでなければなりません。ここで **n** は文字列 **s** の長さです。空白文字あるいは復改文字は入力フィールドを終了させます。ヌル文字が自動的に文字列の最後に付加され，配列の最後の要素に格納されます。

%[探索集合]変換

大カッコ[]で囲まれた文字は，型指定文字 **s** に置き換えられます。アドレス引数は，文字配列を指すポインタです（char *arg*[]）。
カッコの中には，入力文字列を構成する文字からなる探索集合を定義する文字を置きます。
カッコの中の最初にキャレット（ ）を置くと，探索集合は，指定されている文字以外の ASCII 文字（反転探索集合）になります（通常，最初のキャレットの後にキャレットが指定されていなければ，探索集合にはキャレットも含まれます)。
入力フィールドは，ホワイトスペースで区切られていない文字列です。... **scanf** 関数は対応する入力フィールドを，探索集合(あるいは反転探索集合)に含まれない文字に達するまで読みます。この変換の例を2つ示します。

```
%[abcd]     入力フィールドの中で文字 a,b,c,d を探索する
%[^abcd]    入力フィールドの中で a,b,c,d 以外の文字を探索する
```

また，ある範囲の文字を探索集合として簡単に使える範囲機能を使うことができます。たとえばすべての10進数を探索する次の変換指定は，

```
%[0123456789]
```

より簡単に次のように書くことができます。

```
%[0-9]
```

英数文字に関しては，以下のような指定が可能です。

%[A-Z]	すべての英大文字
%[0-9A-Za-z]	すべての数字と英大文字と英小文字
%[A-FT-Z]	英大文字のA～FとT～Z

これらの変換ルールは簡単です。

■ダッシュの前の文字は，後の文字よりもASCIIコードが大きい文字でなければいけません。

■ダッシュは，文字列の先頭または最後にあってはいけません。もし先頭か最後がダッシュならば範囲指定ではなくダッシュ文字そのものを探索します。

■ダッシュの前後に指定する文字は，他の範囲の一部であってはいけません。

以下は範囲指定にはならない例です。

%[-+*/]	四則演算子
%[z-a]	z，-，a の探索
%[+0-9-A-F]	+，-，0～9 と A～F
%[+0-9A-F-]	+，-，0～9 と A～F
%[^-0-9+A-F]	+，-，0～9，A～F 以外の ASCII文字

%e, %E, %f, %g, %G（浮動小数点）変換

入力フィールドにおける浮動小数点数は次に示すような形式をとっていなければなりません。

```
[+/-]dddddddd[.]dddd[E|e][+/-]ddd
```

ここで[]で囲まれた部分はオプションであり，*ddd* は10進，8進，16進いずれかの表記の数です。

%d, %i, %o, %x, %D, %I, %O, %X, %c, %n 変換

unsigned char, unsigned int, unsigned long を指すポインタは, char, int, long を指すポインタが許される変換ならば, どこでも使用できます。

代入抑制文字:

代入抑制文字はアスタリスク（*）です。C の間接参照（ポインタ）演算子もアスタリスクですから, 間違えないでください。
書式指定の中で%の後に*があると, 次の入力フィールドはスキャンされますが, 次のアドレス引数には割り当てられません。スキップされる入力データは, *の後の型指定文字が示す型であるとみなされます。

入力幅指定子:

入力幅指定子(n は10進数)は, 現在の入力フィールドから読まれるべき文字数の上限を指定するものです。
入力フィールドが n 文字より小さい場合は, ...**scanf** 関数はそれらのすべての文字を読み込み, 次のフィールド, 次の書式指定へと移ります。
n 文字読む前に, ホワイトスペースや変換できない文字が現れた場合は, そこまでの文字が読み込まれ, 変換され, 格納されます。その後また関数は次の書式指定に移っていきます。
変換できない文字というのは, 与えられた書式にしたがって変換することのできない文字(たとえば, 書式が8進のときの8や9, 書式が16進のときの J や K) のことです。

入力幅指定子	入力幅はどのような影響を受けるか
n	**n** 文字まで読み込んで変換され, 現在のアドレス引数に格納される。

入力サイズおよび引数型修飾子：

入力サイズ修飾子（**N** と **F**）および引数型修飾子（**h**, **l** および **L**）は，... **scanf** 関数が対応するアドレス引数 *arg*[f] をどう解釈するかに影響を与えるものです。

F と **N** は，*arg* のデフォルトサイズや宣言されたサイズに優先します。

h, **l** および **L** は，続く入力データにどの型（バージョン）が使用されるのか（**h** = short, **l** = long, **L** = long double）を示します。入力データは指定のバージョンに変換され，その入力データ対する *arg* は対応するサイズのオブジェクトを指すことになります（% **h** の場合は short のオブジェクト，% **l** の場合は long か double のオブジェクト，% **L** の場合は long double のオブジェクト）。

修飾子	変換はどのような影響を受けるか
F	デフォルトあるいは宣言されたサイズが変更され，*arg* は far ポインタとして扱われる。
N	デフォルトあるいは宣言されたサイズが変更され，*arg* は near ポインタして扱われる。ヒュージモデルでは，どの変換とも一緒には使用できない。
h	**d,i,o,u,x** 型：入力を short int に変換し short 型オブジェクトに格納する。 **D,I,O,U,X** 型：影響なし。 **e,f,c,s,n,p** 型：影響なし。
l	**d,i,o,u,x** 型：入力を long int に変換し，long 型オブジェクトに格納する。 **e,f** 型：入力を double に変換し，double 型オブジェクトに格納する。 **D,I,O,U,X** 型：影響なし。 **c,s,n,p** 型：影響なし。
L	**e,f,g** 型：入力を long double に変換し，long double 型オブジェクトに格納する。

scanfがスキャンをやめる場合：

scanf 関数は，通常のフィールド終了文字（ホワイトスペース）を読む前に，さまざまな理由により，あるフィールドのスキャンをやめる場合があります。

scanf 関数は，次に示すことが起きた場合には，現在のフィールドのスキャンおよび格納をやめて，次の入力フィールドへ移ります。

■代入抑制文字（*）が書式指定の中で%の後に指定されている場合。
　現在の入力フィールドはスキャンされますが，値は格納されません。
■入力幅指定子で指定されている個数の文字を読み込んだ場合。
■現在の書式で変換できない文字を読んだ場合（たとえば10進形式でのA）
■入力フィールドの次の文字が探索リストに入っていない場合。

これらの理由から **scanf** 関数が現在の入力フィールドのスキャンをやめたときには，次の文字は読み込まれなかったものとし，次の入力フィールドの先頭の文字となるか，あるいは次の読み込み操作の先頭の文字となります。

scanf は次の場合に終了します。

■入力フィールドの次の文字が，書式文字列の中の非ホワイトスペースと矛盾するとき。
■入力フィールドの次の文字がEOFのとき。
■書式文字列がすべて使い果たされたとき。

書式文字列中に，書式指定の一部分でない文字列が現われた場合は，入力フィールドの現在の文字の並びと一致しなければなりません。**scanf** は，一致した文字をスキャンしますが格納はしません。一致しない文字が現われたときは，その文字はあたかも読まれなかったように，入力フィールドにとどまったままになります。

戻り値　**scanf** は，正しくスキャンし，変換し，格納した入力フィールドの数を返します。戻り値には，値を格納しなかった入力フィールドの数は含まれません。

ファイルエンドを読み込んだ場合は，戻り値は EOF になります。

値を格納したフィールドがなかった場合は，戻り値は0となります。

可搬性　**scanf** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**cscanf, fscanf, printf, sscanf, vfscanf, vsscanf**

searchpath

機能　DOSのパスを探索してファイルを探します。

形式　char * searchpath(const char * *file*) ;

プロトタイプ　dir.h

解説　**searchpath**は，*file*で与えられたファイルを，DOSのパスを使って探します。DOSのパスは，PATH=...という形式の文字列です。そのファイルのフルパス名の文字列を指すポインタが関数値として返されます。
searchpathは，最初にカレントドライブのカレントディレクトリをチェック探します。ファイルがそこで見つからない場合は，環境変数PATHを取り出して，その中の各ディレクトリの中で探索を行ないます。ファイルが見つかるか，パス名を使い果たすまで探索を続けます。
ファイルが見つかると，そのフルパス名を含む文字列が返されます。この文字列は，そのファイルをアクセスする**fopen**や**exec...**などの呼び出しで使用することができます。
この文字列は，静的バッファの中に格納されるので，次の**searchpath**の呼び出しによって上書きされます。

戻り値　ファイルが見つかった場合は，ファイル名を含む文字列を指すポインタを返します。そうでない場合はNULLを返します。

可搬性　**searchpath**はMS-DOSに特有の関数です。

関連項目　**exec..., spawn..., system**

例

```
#include <stdio.h>
#include <dir.h>

main()
{
   char *p;

   p = searchpath("TLINK.EXE");
   printf("Search for TLINK.EXE : %s¥n", p);
   p = searchpath("NOTEXIST.FIL");
   printf("Search for NOTEXIST.FIL : %s¥n", p);
}
```

プログラム出力

```
Search for TLINK.EXE : C:¥BIN¥TLINK.EXE
Search for NOTEXIST.FIL : (null)
```

segread

機能　セグメントレジスタを読み出します。

形式　# include <dos.h>

void segread(struct SREGS * *segp*) ;

プロトタイプ　dos.h

解説　**segread** は，セグメントレジスタの現在の値を，*segp* が指す構造体の中に格納します。これは，**intdosx** や **int86x** とともに使用する目的で呼び出されます。

戻り値　ありません。

可搬性　**segread** は80x86ファミリィのプロセッサに特有の関数です。

関連項目　**FP_OFF, intdos, int86, MK_FP, movedata**

setblock

機能　以前に割り当てられたブロックの大きさを変更します。

形式　int setblock(unsigned *segx*, unsigned *newsize*);

プロトタイプ　dos.h

解説　**setblock** は，メモリセグメントの大きさを変更します。*segx* には，以前の **allocmem** の呼び出しで返されたセグメントアドレスを指定します。*newsize* には，新たに要求するメモリ量をパラグラフ単位で与えます。

戻り値　**setblock** は，成功した場合には-1を返します。エラーの場合は確保可能な最大のブロックサイズを返し，***_doserrno*** をセットします。

可搬性　**setblock** は MS-DOS に特有の関数です。

関連項目　**allocmem**

setbuf

機能 ストリームにバッファリングを割り当てます。

形式 # include <stdio.h>

void setbuf(FILE ＊ *stream*, char ＊ *buf*)；

プロトタイプ stdio.h

解説 **setbuf** は，自動的に割り当てられたバッファにかえて，バッファ *buf* を I/O バッファリングに使用できるようにします。このバッファは，*stream* をオープンした後で使用されます。

buf が NULL の場合は，I/O はバッファリングされません。それ以外の場合は完全にバッファリングされます。バッファの大きさは BUFSIZ(stdio.h の中で定義されている）バイトなければなりません。

stdin と *stdout* は，リダイレクトされていなければバッファリングされません。そうでない場合は完全にバッファリングされます。**setbuf** は，使用されるバッファリングの形式を変更するためにも使えます。

バッファリングをしない場合，ストリームに文字を書き出すと，すぐにファイルやデバイスに出力されます。バッファリングする場合は，文字はバッファに蓄えられ，ブロック単位で書かれていきます。

setbuf は，*stream* をオープンした直後，あるいは **fseek** を呼び出した直後以外に呼び出すと予期できない結果が生じます。*stream* をバッファリングしないと指定した後で **setbuf** を呼び出すのは正当であり問題は起こりません。

よく起こるエラーの原因として，バッファを自動（ローカル）変数に割り当ててしまうことがあります。バッファが宣言された関数から戻る前に，ファイルをクローズするのを忘れてしまうことが多いのです。

戻り値 ありません。

可搬性 **setbuf** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**fflush**，**fopen**，**fseek**，**setvbuf**

例　　**setvbuf** を参照してください。

setcbrk

機能　コントロールブレークの設定をセットします。

形式　int setcbrk(int *cbrkvalue*)；

プロトタイプ　dos.h

解説　**setcbrk** は，DOS システムコール0x33を使って，コントロールブレークチェックをオンまたはオフに設定します。

cbrkvalue = 0	オフにする(コンソール，プリンタ，通信デバイスのI/Oにおいてのみチェックを行ないます)。
cbrkvalue = 1	オンにする(すべてのシステムコールにおいてチェックを行ないます)。

戻り値　**setcbrk** は，渡された値 *cbrkvalue* を返します。

可搬性　**setcbrk** は MS-DOS に特有の関数です。

関連項目　**getcbrk**

setdate

機能 DOSの日付をセットします。

形式 #include <dos.h>

void setdate(struct date * *datep*);

プロトタイプ dos.h

解説 **setdate**は, *datep*が指す**date**構造体の中に入っている日付を, システムの日付（月, 日, 年）にセットします。

date構造体は次のように定義されています。

```
struct date {
   int da_year;        /* 年 */
   char da_day;        /* 日 */
   char da_mon;        /* 月 */
};
```

戻り値 ありません。

可搬性 **setdate**はMS-DOSに特有の関数です。

関連項目 **getdate**, **gettime**, **settime**

例 **getdate**を参照してください。

setdisk

機能　カレントディスクドライブをセットします。

形式　int setdisk(int *drive*)；

プロトタイプ　dir.h

解説　**setdisk** は，カレントドライブを *drive* に指定されたものにします（0=A，1=B，2=C）。DOS システムコール0x0E と同じです。

戻り値　**setdisk** は，利用可能なドライブの総数を返します。

可搬性　**setdisk** は MS-DOS に特有な関数です。

関連項目　**getdisk**

setdta

機能　　ディスク転送アドレスをセットします。

形式　　void setdta(char far * *dta*)；

プロトタイプ　dos.h

解説　　**setdta** は，ディスク転送アドレス（DTA）の現在の設定を，*dta* に指定された値に変更します。

戻り値　　ありません。

可搬性　　**setdta** は MS-DOS に特有のものです。

関連項目　　**getdta**

setftime

機能　ファイルの日付と時刻をセットします。

形式　# include <io.h>

int setftime(int *handle*, struct ftime ∗ *ftimep*) ;

プロトタイプ　io.h

解説　**setftime** は，オープンされている *handle* に結びつけられているディスクファイルのファイル時刻・日付に，*ftimep* が指す **ftime** 構造体に格納されている日付・時刻をセットします。

ftime 構造体は次のように定義されています。

```
struct ftime {
   unsigned ft_tsec: 5;      /* 秒(2秒単位) */
   unsigned ft_min: 6;       /* 分          */
   unsigned ft_hour: 5;      /* 時          */
   unsigned ft_day: 5;       /* 日          */
   unsigned ft_month: 4;     /* 月          */
   unsigned ft_year: 7;      /* 年−1980     */
};
```

戻り値　成功した場合，**setftime** は0を返します。

エラーの場合は-1を返し，*errno* に次のいずれかをセットします。

EINFNC	無効なファンクション番号
EBADF	ファイル番号が正しくない

可搬性　**setftime** は MS-DOS に特有の関数です。

関連項目　**getftime**

例　**getdate** を参照してください。

setjmp

機能　ノンローカル goto のためのセットアップを行ないます。

形式

```
#include <setjmp.h>
int setjmp(jmp_buf jmpb);
```

プロトタイプ　setjmp.h

解説　**setjmp** は，現在の完全なタスクの状態をとらえて *jmpb* に格納し，0を返します。

setjmp を行なった後で，その *jmpb* を引数として **longjmp** を呼び出すと，*jmpb* に入っていたタスクの状態が回復され，**setjmp** が値 *val* を返したかのようにリターンします。

タスクの状態とは以下のものをいいます。

- ■すべてのセグメントレジスタ（CS，DS，ES，SS）
- ■レジスタ変数（SI，DI）
- ■スタックポインタ（SP）
- ■フレームベースポインタ（BP）
- ■フラグ

タスクの状態は，コルーチンを実現するために **setjmp** が使用されるのに充分なものです。

setjmp は，**longjmp** より前に呼び出しておかなければなりません。**setjmp** を呼んで *jmpb* を設定したルーチンは，**longjmp** が呼び出されるまではアクティブなままでなければならず，リターンすることはできません。もし，そうしてしまった場合は，結果は予測できません。

setjmp は，プログラムの低レベルサブルーチンにおいて起こるエラーや例外を処理する際に便利なものです。

戻り値　**setjmp** は，最初に呼び出されたときは0を返します。

可搬性　　**setjmp** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**longjmp**，**signal**

例　　**longjmp** を参照してください。

setmem

機能 ある範囲のメモリに値を代入します。

形式 void setmem(void * *dest*, unsigned *length*, char *value*) ;

プロトタイプ mem.h

解説 **setmem** は, *dest* が指すブロックの先頭から *length* バイトを, *value* で埋めます。

戻り値 ありません。

可搬性 **setmem** は8086ファミリィに特有の関数です。

関連項目 **memset**, **strset**

setmode

機能 オープンファイルのモードをセットします。

形式 # include <fcntl.h>

int setmode(int *handle*, int *amode*);

プロトタイプ io.h

解説 **setmode** は，*handle* に結びつけられているオープンファイルのモードを，バイナリかテキストにセットします。引数 *amode* は，O_BINARY か O_TEXT のどちらかの値でなければなりません（これらのシンボリック定数は fcntl.h の中で定義されています）。

戻り値 **setmode** は，成功した場合は0を返します。エラーの場合は-1を返し，***errno*** に ENIVAL（引数が正しくない）をセットします。

可搬性 **setmode** は UNIX システムで使用できます

関連項目 **_creat, creat, _open, open**

settime

機能　システム時刻をセットします。

形式　# include <dos.h>

void settime(struct time * *timep*) ;

プロトタイプ　dos.h

解説　**settime** は，*timep* が指す **time** 構造体の中に入っている時刻をシステムの時刻にセットします。

time 構造体は次のように定義されています。

```
struct time {
   unsigned char ti_min;        /* 分      */
   unsigned char ti_hour;       /* 時      */
   unsigned char ti_hund;       /* 1/100秒 */
   unsigned char ti_sec;        /* 秒      */
};
```

戻り値　ありません。

可搬性　**settime** は MS-DOS に特有の関数です。

関連項目　**ctime**, **getdate**, **gettime**, **setdate**, **time**

setvbuf

機能　ストリームにバッファリングを割り当てます。

形式　# include <stdio.h>

int setvbuf(FILE * *stream*, char * *buf*, int *type*, size_t *size*);

プロトタイプ　stdio.h

解説

setvbuf は，自動的に割り当てられるバッファにかえて，バッファ *buf* をI/O バッファリングに使用できるようにします。このバッファは，stream をオープンした後で使用できるようになります。

buf が NULL の場合は，バッファは **malloc** を使って確保されます。バッファに確保される領域の大きさとして *size* が使われます。引数 *size* はバッファの大きさを指定するものであり，ゼロより大きくなければなりません。

注意：*size* の最大値は32767に制限されます。

stdin と *stdout* は，リダイレクトされていなければバッファリングされません。そうでない場合は完全にバッファリングされます。

バッファリングをしない場合，ストリームに文字を書き出すと，すぐにファイルやデバイスに出力されます。バッファリングする場合は，文字はバッファに蓄えられ，ブロック単位で書かれていきます。

引数 *type* は次のいずれかです。

_IOFBF　ファイルは完全にバッファリングされます。つまり，入力操作はバッファがいっぱいになるまで行なわれます。出力では，バッファが完全にいっぱいになるまでファイルに書き出されません。

_IOLBF　ファイルは行バッファリングされます。入力操作は，上と同様にバッファがいっぱいになるまで行なわれます。出力では，復改文字が書かれたときにバッファがフラッシュされます。

_IONBF　ファイルはバッファリングされません。*buf* と *size* 引数は無視されます。各入力操作はファイルから直接行なわれます。出力も，データが書かれるとすぐにファイルに書き出されます。

よく起こるエラーの原因として，バッファを自動（ローカル）変数に割り当ててしまうことがあります。バッファが宣言された関数から戻る前に，ファイルをクローズするのを忘れてしまうことが多いのです。

戻り値　**setvbuf** は，成功した場合は0を返します。*type* と *size* に正しくない値を与えた場合，およびバッファを確保するための充分な領域がなかった場合には，0以外の値を返します。

可搬性　**setvbuf** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**fflush**, **fopen**, **setbuf**

例

```
#include <stdio.h>

main()
{
   FILE *input, *output;
   char bufr[512];

   input = fopen("file.in", "r");
   output = fopen("file.out", "w");

   /* 最小限のディスクアクセス用のストリームをセットアップ.
      専用の文字バッファを使う.                                */
   if (setvbuf(input, bufr, _IOFBF, 512) != 0)
      printf("failed to set up buffer for input file¥n");
   else
      printf("buffer set up for input file¥n");

   /* 出力ストリームをセットアップ.mallocを間接的に呼び出して
      確保した領域を使って行バッファリングを行なう.          */
   if (setvbuf(output, NULL, _IOLBF, 132) != 0)
      printf("failed to set up buffer for output file¥n");
   else
      printf("buffer set up for output file¥n");

   /* ここでファイルI/Oを行なう */

   /* ファイルをクローズする */
   fclose(input);
   fclose(output);
}
```

setvect

機能　割り込みベクタエントリをセットします。

形式　void setvect(int *interruptno*, void interrupt (* *isr*)());

プロトタイプ　dos.h

解説　8086ファミリィのすべてのプロセッサは割り込みベクタのセットを持っており，これには0～255の番号がふられています。各ベクタの中の4バイト値は，割り込み関数が置かれている場所を示すアドレスです。
setvect は, *intrruptno* で指定されたベクタの値に, 新しい値 *isr* をセットします。*isr* は, 新しい割り込み関数のアドレスを含む far ポインタです。C のルーチンを interrupt ルーチンと宣言している場合には，*isr* に渡せるのはそのルーチンのアドレスだけになります。

注意：dos.h で宣言されているプロトタイプを使えば，どのメモリモデルにおいても，割り込み関数のアドレスを **setvect** に渡すことができます。

戻り値　ありません。

可搬性　**setvect** は8086ファミリィのプロセッサに特有の関数です。

関連項目　**getvect**

setverify

機能　ベリファイフラグの状態をセットします。

形式　void setverify(int *value*);

プロトタイプ　dos.h

解説　**setverify** は，ベリファイフラグの状態を *value* にセットします。

value = 0　ベリファイフラグオフ
value = 1　ベリファイフラグオン

ベリファイフラグはディスクへの出力を制御します。ベリファイがオフのときには書き込みの照合（ベリファイ）は行なわれず，ベリファイがオンのときはすべてのディスクへの書き込みにおいて，データが適切に書き出されたかどうかを確かめるために照合が行なわれます。

戻り値　ありません。

可搬性　**setverify** は MS-DOS に特有の関数です。

関連項目　**getverify**

signal

機能　シグナル処理操作を指定します。

形式　# include <signal.h>

void (* signal(int *sig*, void(* *func*)(int *sig*[, int *subcode*])))(int);

プロトタイプ　signal.h

解説　**signal** は，シグナル番号 *sig* が受け取られた後でどのように処理されるかを決定します。ユーザ指定のハンドラルーチンをインストールすることもでき，signal.h の中であらかじめ定義されている2個のハンドラのいずれかを使うこともできます。

2つの定義済みハンドラは次の通りです。

関数ポインタ	意味
SIG_DFL	プログラムを終了する。
SIG_IGN	この型のシグナルは無視する。

signal.h の中で定義されている3番目の定義済みハンドラは SIG_ERR です。これは，**signal** からのエラー戻りを示すために使われます。

シグナル型およびそのデフォルト値は次の通りです。

シグナル型	意味
SIGABRT	異常終了。デフォルト動作は**_exit**(3)の呼び出しと同じ。
SIGFPE	ゼロ除算などの正しくない操作によって引き起こされた数学的なエラー。デフォルト動作は**_exit**(1)の呼び出しと同じ。
SIGILL	不当な操作。デフォルト動作は**_exit**(1)の呼び出しと同じ。
SIGINT	***CTRL-C*** 割り込み。デフォルト動作はINT 23Hの実行。
SIGSEGV	主記憶への不当なアクセス。デフォルト動作は**_exit**(1)の呼び出しと同じ。
SIGTERM	プログラム終了要求。デフォルト動作は**_exit**(1)の呼び出しと同じ。

signal.hは，*sig_atomic_t* という型を定義しています。これは，非同期の割り込みの存在下でアトム的にプロセッサがロードやセーブすることができる最大の整数型です（8086ファミリィでは，これは16ビットワード，すなわちTurbo Cの整数)。
raise 関数，あるいは外部イベントによってシグナルが生成された場合，以下のことが行なわれます。

1. そのシグナルに対してユーザ指定のハンドラがインストールされていれば，そのシグナル型に対する動作がSIG_DFLにセットされます。
2. ユーザ指定の関数が，シグナル型を引数として呼び出されます。

ユーザ指定のハンドラ関数は，return 文や **abort**, **_exit**, **exit**, **longjmp** の呼び出しで終了することができます。

Turbo C は，シグナル型が SIGFPE，SIGSEGV，SIGILL の場合には，ANSI C への拡張をインプリメントしています。ユーザ指定のハンドラ関数は，引数を1つあるいは2つ追加して呼び出されます。**raise** 関数を明示的に呼び出した結果として，SIGFPE, SIGSEGV, SIGILL のいずれかが発行された場合，ユーザ指定のハンドラ関数は，そのハンドラが明示的に呼び出されていることを示す整数値を追加引数として呼び出されます。SIG-FPE, SIGSEGV, SIGILL に対する明示的呼び出しの値は次のようになっています（float.h の中の宣言を参照してください）。

シグナル型	**起動値**
SIGFPE	FPE_EXPLICITGEN
SIGSEGV	SEGV_EXPLICITGEN
SIGILL	ILL_EXPLICITGEN

SIGFPE が浮動小数点例外によって発行された場合，ユーザハンドラは，そのシグナルの FPE_*xxx* 型を示す追加引数をともなって呼び出されます。プロセッサ例外の結果として，SIGSEGV，SIGILL，あるいは SIGFPE シグナルの変種(FPE_INTOVFLOW または FPE_INTDIV0) が引き起こされた場合には，ユーザハンドラは2つの追加引数をともなって呼び出されます。

1. SIGFPE，SIGSEGV，あるいは SIGILL の例外型(これらの型については float.h を参照してください)。この最初の引数は通常の ANSI のシグナル型です。
2. ユーザ指定のハンドラを呼び出した割り込みハンドラのスタックの中を指す整数ポインタ。このポインタは，例外が発生したときに退避されたプロセッサレジスタのリストを指します。レジスタは，割り込み関数への引数と同じ順序で並んでいます。つまり，BP, DI, SI, DS, ES, DX, CX, BX, AX, IP, CS, FLAGS の順です。ハンドラがリターンするときにレジスタの値を変更したい場合は，このリストにおける対応する部分を変更します。たとえば，SI に新しい値をセットしたい場合は次のようになります。

```
*((int *)list_pointer + 2) = new_SI_value;
```

このようにして，ハンドラはレジスタの値を調べたり，新しい値に変更することができます（後に示す**例 2** を参照してください)。

次に示す SIGFPE 型のシグナルが起こり得ます（あるいは生成され得ます)。これらは，メイン CPU における"INTEGER DIVIDE BY ZERO", "INTERRUPT ON OVERFLOW"に加えて，80x87が検出することができる例外にも対応しています。これらの宣言は float.h の中にあります。

SIGFPE シグナル	**意味**
FPE_INTOVFLOW	OF フラグをセットして INTO 命令を実行
FPE_INTDIV0	ゼロによる整数の除算
FPE_INVALID	不当な操作
FPE_ZERODIVIDE	ゼロによる除算
FPE_OVERFLOW	数値オーバーフロー
FPE_UNDERFLOW	数値アンダーフロー
FPE_INEXACT	精度
FPE_EXPLICITGEN	ユーザプログラムが **raise**(SIGFPE) を実行

注意：FPE_INTOVFLOW と FPE_INTDIV0シグナルは，整数に対する操作によって生成されます。その他は浮動小数点に対する操作によって生成されます。浮動小数点例外が生成されるかどうかは，コプロセッサ制御ワードに依存します。コプロセッサ制御ワードは **_control87** を使って変更することができます。デノーマル（不正規化数）例外は Turbo C によって処理され，シグナルハンドラには渡されません。

次の SIGSEGV 型のシグナルが起こり得ます。

SIGSEGV シグナル	意味
SEGV_BOUND	bound コンストレイント例外
SEGV_EXPLICITGEN	**raise**(SIGSEGV)が実行された

注意：8088および8086プロセッサは bound 命令を備えていません。80186/286/386，および NEC の V シリーズのプロセッサはこの命令を持っています。したがって，8088および8086プロセッサでは，SIGSEGV シグナルの SEGV_BOUND 型は起こり得ません。Turbo C は bound 命令を生成しませんが，インラインコードや独立にアセンブルされリンクされるアセンブリ言語ルーチンの中では使用され得ます。

次の SIGILL 型のシグナルが起こり得ます。

SIGILL シグナル	意味
ILL_EXECUTION	不当な操作が試みられた
ILL_EXPICITGEN	**raise**(SIGILL)が実行された

注意：8088，8086，V20，V30プロセッサは不当操作例外をもっていません。186，286，386，V40，V50プロセッサはこの例外型を持っています。したがって，8088，8086，V20，V30プロセッサでは SIGILL シグナルの ILL EXECUTION 型は起こり得ません。

注意：シグナル型が SIGFPE, SIGSEGV, SIGILL の場合は，8087の状態が正しくなくなっている，整数除算の結果が間違っている，オーバーフローすべきでない操作が行われた，bound 命令が失敗した，不当な操作が行なわれたなどのいずれかであるので，シグナルハンドラから戻ることは一般に望ましいものではありません。唯一の例外は，ハンドラがレジスタを変更して理にかなったリターンの状況が作り出されるか，シグナル型がそのシグナルは明示的に生成されたものであることを示している場合です（例：FPE_EXPLICITGEN, SEGV_EXPLICITGEN, ILL_EXPLICITGEN)。一般にこのような場合は，エラーメッセージを表示し，**exit**, **exit**, **abort** を使用して，プログラムを終了させるのが良いでしょう。そうでない場合は，プログラム続行の際にどのような事態になるかは予想ができません。

戻り値　呼び出しが成功した場合，**signal** は，指定されたシグナル型に対するハンドラルーチンを指すポインタを返します。呼び出しが失敗した場合は SIG ERR を返し，***errno*** に EINVAL をセットします。

可搬性　**signal** は ANSI C と互換性があります。

関連項目　**abort**, **_control87**, **ctrlbrk**, **exit**, **longjump**, **raise**, **setjmp**

例 1

```
/* この例では，シグナルハンドラルーチンを，CTRL-C が押されたときに
   実行されるようにインストールします．  */

#include <stdio.h>
#include <signal.h>

void Catcher(int sig)
{
    printf("¥nNow in break routine¥n");
    exit(1);
}

main()
{
    signal(SIGINT, Catcher);
    for(;;)
        printf("¥nIn main() program¥n");
}
```

例2

```
/* この例では，SIGFPE のためのシグナルハンドラルーチンをインストールし，
   整数のオーバーフロー条件をとらえ，調整値を AX にセットして，
   リターンします． */

#pragma inline
#include <stdio.h>
#include <signal.h>

void Catcher(int sig, int type, int *reglist)
{
    printf("Caught it!¥n");
    *(reglist+8) = 3;          /* AX に 3 を返す */
}

main()
{
    signal(SIGFPE, Catcher);
    asm    mov    ax, 07FFFH          /* AX = 32767 */
    asm    inc    ax                  /* オーバーフローを起こす */
    asm    into                       /* ハンドラをアクティブにする */

    /* ハンドラは AX に 3 を返す．それが起こらなければ，dec 命令の後の
       into が実行されたときに，他の例外が発生する */

    asm    dec    ax                  /* オーバーフローなし */
    asm    into                       /* ハンドラはアクティブにならない */
}
```

sin

機能　　正弦を求めます。

形式　　# include <math.h>

double sin(double *x)*;

プロトタイプ　math.h

解説　　**sin** は，入力値の正弦（サイン）を計算します。角度 x はラジアン単位で指定します。

戻り値　　**sin** は，入力値の正弦を-1～1の範囲で返します。

sin のエラー処理は，**matherr** を使って変更することができます。

可搬性　　**sin** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　　**acos, asin, atan, atan2, cos, cosh, tan, tanh**

sinh

機能　双曲線正弦を求めます。

形式　# include <math.h>

double sinh(double *x*)；

プロトタイプ　math.h

解説　**sinh** は，実数の引数の双曲線正弦（ハイパボリックサイン）を計算します。

戻り値　**sinh** は，*x* の双曲線正弦を返します。

計算結果がオーバーフローする場合は，**sinh** は適切な符号の HUGE_VAL を返します。

sinh のエラー処理は，**matherr** 関数で変更することができます。

可搬性　**sinh** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**acos, asin, atan, atan2, cos, cosh, sin, tan, tanh**

sleep

機能　実行を一定時間停止します。

形式　void sleep(unsigned *seconds*);

プロトタイプ　dos.h

解説　**sleep** を呼び出すと，現在実行中のプログラムは，引数 *seconds* に指定された秒数だけ実行を停止します。停止時間は，PC-9801ではDOSのクロック，IBM PCでは1/100秒単位の近似値になります。

戻り値　ありません。

可搬性　**sleep** は UNIX システムで使用できます。

関連項目　**delay**

sopen

機能　シェアドファイルをオープンします。

形式　# include <fcntl.h>
# include <sys￥stat.h>
# include <share.h>
# include <io.h>
int sopen(char * *path*, int *access*, int *shflag*, int *mode*);

プロトタイプ　io.h

解説　**sopen** は，*path* に指定されたファイルを，*access*, *shflag*, および *mode* の値にしたがって，読み出し，書き込み，あるいは両用のシェアドファイルとしてオープンします。
sopen は，マクロとして次のように定義されています。

```
open(path, (access) | (shflag), mode)
```

引数 *access* は，次に示す2つのリストのフラグをビットごとに OR することによって作ります。**リスト 1** からは1つのフラグしか選べません。**リスト 2** では，複数のフラグを組み合わせることができます。

リスト1：読み出し/書き込みフラグ

O_RDONLY	読み出し専用としてオープン
O_WRONLY	書き込み専用としてオープン
O_RDWR	読み出しおよび書き込み用としてオープン

リスト2：他のアクセスフラグ

O_NDELAY	使用されません。UNIXとのコンパチビリティを図るためのものです。
O_APPEND	これがセットされていると，ファイルポインタはまずファイルの終わりに位置づけられます。
O_CREAT	指定のファイルがすでに存在していれば，このフラグは意味を持ちません。存在していなければ，ファイルは新規に作成され，*mode*のビットは**chmod**の場合と同様にして，属性ビットにセットされます。
O_TRUNC	ファイルが存在している場合は，その長さは0に削られます。ファイル属性はそのままで変化しません。
O_EXCL	使用されません。UNIXとのコンパチビリティを図るためのものです。
O_BINARY	このフラグは，明示的にバイナリモードでファイルをオープンすることを示すのに使用されます。
O_TEXT	このフラグは，明示的にテキストモードでファイルをオープンすることを示すのに使用されます。

これらのO_...シンボリック定数は，fcntl.hの中で定義されています。O_BINARYもO_TEXTも与えられていない場合は，ファイルは，グローバル変数***_fmode***にセットされている変換モードでオープンされます。O_CREATEフラグが*access*を作るために使われている場合は，引数*mode*を次に示すシンボリック定数から作る必要があります。

*mode*の値	**アクセス許可**
S_IWRITE	書き込み可
S_IREAD	読み出し可
S_IREAD | S_IWRITE	読み書き可

引数 *shflag* には，ファイル ***path*** に対して許可されるファイルシェアリングモードを指定します。以下の定数が share.h の中で定義されています。

shflag の値	意味
SH_COMPACT	コンパチブルモードをセットする
SH_DENYRW	読み出し，書き込みアクセスともに不可
SH_DENYWR	書き込みアクセス不可
SH_DENYRD	読み出しアクセス不可
SH_DENYNONE	読み出し，書き込みアクセスともに許可
SH_DENYNO	読み出し，書き込みアクセスともに許可

戻り値

成功すると，**sopen** は負でない整数(ファイルハンドル)を返し，ファイルの現在位置を示すファイルポインタをファイルの先頭に置きます。エラーの場合は-1を返し，***errno*** に次のいずれかの値をセットします。

ENOENT	パス名またはファイル名が見つからなかった
EMFILE	オープンされているファイルが多すぎる
EACCESS	アクセスが拒否された
EINVACC	アクセスコードが正しくない

可搬性

sopen は UNIX システムで使用可能です。UNIX バージョン7では，O_型のニーモニックは定義されていません。UNIX システム III は，O_BINAR-Y を除くすべての O_型ニーモニックを使用しています。

関連項目

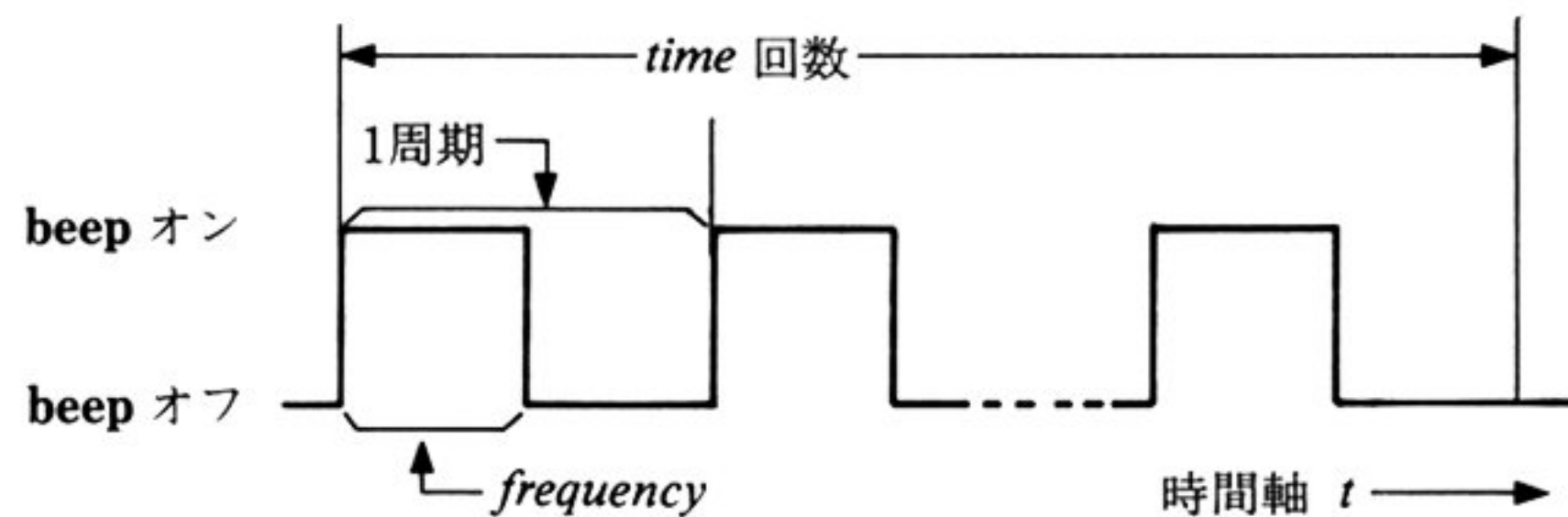
chmod，**close**，**creat**，**lock**，**lseek**，**_open**，**open**，**unlock**

sound (PC-9801)

機能　ビープ音の波形を指定します。

形式　unsigned int sound(unsigned *frequency*);

プロトタイプ　dos.h

機能説明　**sound**は，ビープ音の波の間隔（周期）*frequency*を指定します。実際に音を発生するのは**beep**によって行ないます。
beepは，指定した繰り返し回数*time*だけ波を発生します。このとき，波の間隔*frequency*と，繰り返し回数*time*の関係は次のようになります。

したがって，単位周期＝*frequency*×2となります。
たとえば，定時間で波の間隔（周期）を変えて音を出力したいときは，次の式のαを一定に保つようにしてください。

$$frequency \times time = \alpha$$

下記のプログラムでは，最初に単位周期20で繰り返し回数5000で比較的高い音が発生し，次に単位周期1000で繰り返し回数100で比較的低い音が発生します。ともに音の発生時間は同じです。

```
cyce = 50000;
freq = 10;  sound(freq); beep(cyce/freq);    /* 高い音 */
freq = 500; sound(freq); beep(cyce/freq);    /* 低い音 */
```

また，一番高い音は波の間隔（周期）*frequency*を0にしたときです。

注意 1：frequency は，本来は周波数の意味ですが，PC-9801シリーズでは逆数の意味である周期として扱っています。

注意 2：マシンのタイプにより，また基本クロックの違いにより音色が変化するので，各マシンに合わせて調整してください。

戻り値 **sound** は，直前に設定されていた波の間隔（周期）を返します。

可搬性 この関数は PC-9801シリーズでのみ動作します。

関連項目 **beep**

例

```
#include  <dos.h>

void beepbeep()
{
    unsigned  f;
    f = sound(10);          /* 直前の値を退避する */
    beep(50000 / 10);       /* 高い音             */
    sound(700);
    beep(50000 / 700);      /* 低い音             */
    sound(f);               /* 以前の値を復元する */
}
```

sound (IBM PC)

機能 PC のスピーカを指定の周期で鳴らします。

形式 void sound(unsigned *frequency*);

プロトタイプ dos.h

機能説明 **sound** は，PC のスピーカを指定の周期で鳴らします。*frequency* には音の周期をヘルツ（回数/秒）単位で指定します。**sound** によって鳴り出したスピーカを止めるには，**nosound** を呼び出してください。

戻り値 ありません。

可搬性 **sound** は，IBM PC とその互換機でのみ動作します。Turbo Pascal にも同様なルーチンがあります。

関連項目 **delay**, **nosound**

例

```
/* 7-Hz の音を 10秒間発生させます.

   本当の話： 7-Hz は，鶏の頭蓋骨の中で共鳴する音の周期です.
   これは，オーストラリアのある新しい工場で経験的に得られた
   結果です.  7-Hz の音を発するその工場は養鶏場のすぐ近くに
   位置していましたが，工場の操業が始まると，養鶏場の鶏がす
   べて死んでしまったのです.

   PC によっては，7-Hz の音は発生しないかもしれません.     */

main()
{
     sound(7);
     delay(10000);
     nosound();
}
```

spawn...

機能　子プロセスを生成して実行します。

形式

include <process.h>
include <stdio.h>
int spawnl(int *mode*, char ＊ *path*, char ＊ *arg0*, *arg1*,
　　　　..., *argn*, NULL)；
int spawnle(int *mode*, char ＊ *path*, char ＊ *arg0*, *arg1*,
　　　　..., *argn*, NULL, char ＊ *envp*[])；
int spawnlp(int *mode*, char ＊ *path*, char ＊ *arg0*, *arg1*,
　　　　..., *argn*, NULL)；
int spawnlpe(int *mode*, char ＊ *path*, char ＊ *arg0*, *arg1*,
　　　　..., *argn*, NULL,char ＊ *envp*[])；
int spawnv(int *mode*, char ＊ *path*, char ＊ *argv*[])；
int spawnve(int *mode*, char ＊ *path*, char ＊ *argv*[], char ＊ *envp*[])；
int spawnvp(int *mode*, char ＊ *path*, char ＊ *argv*[])；
int spawnvpe(int *mode*, char ＊ *path*, char ＊ *argv*[], char ＊ *envp*[])；

プロトタイプ　process.h

解説

spawn...ファミリィの関数は，子プロセスを生成して実行します。子プロセスをロードし，実行するのに充分なメモリが存在していなければなりません。

mode の値は，**spawn...**を呼び出した後，呼び出し関数（親プロセス）がとる動作を指定するものです。*mode* に指定する値は以下の3つです。

P_WAIT	子プロセスが実行を終了するまで親プロセスを一時停止状態にする。
P_NOWAIT	子プロセスの実行と平行して親プロセスも実行を続ける。
P_OVERLAY	親プロセスが占めていたメモリ領域に子プロセスをオーバーレイする。**exec...**呼び出しと同じ。

注意：P_NOWAIT は現在使用できません。使用するとエラー値を発生します。

path は，呼び出される子プロセスのファイル名です。**spawn...**関数を呼び出すと，DOS の標準探索アルゴリズムを使って *path* が探されます。

- 拡張子もピリオドもないとき：その名前のファイルを探します。見つからない場合は，.COM を付加してもう一度探します。それでも見つからなければ，.EXE を付加してもう一度探します。
- 拡張子が与えられているとき：その名前のファイルだけを探します。
- ピリオドが与えられているとき：拡張子のないそのファイルだけを探します。
- **spawn...**関数にサフィックス **p** がついている場合，*path* にディレクトリが明示されていなければ，まずカレントディレクトリを探し，次に DOS の環境変数 PATH に指定されているディレクトリを探します。

ファミリィ名 **spawn** の後につけるサフィックス（接尾辞）**l,v,p,e** は，その関数がどのように動作するかを示しています。

p　DOS の環境変数 PATH に指定されているディレクトリの中で子プロセスを探します。ただし，*path* にディレクトリ名が含まれている場合はそこを先に探し，次に環境変数 PATH を調べます。*path* にディレクトリ名が含まれていない場合は，カレントディレクトリを先に探します。**p** がつかない場合は，ルートとカレントディレクトリしか探されません。

l　引数ポインタ *arg0*, *arg1*, ..., *argn* は，1つずつ独立した引数として渡されることを意味します。**l** は通常，渡す引数の数があらかじめわかっているときに指定します。

v　引数ポインタ *argv*[0]，. . . ，*argv*[*n*]は，ポインタの配列として渡されることを意味します。**v** は通常，可変個の引数を渡すときに指定します。

e　引数 *envp* を子プロセスに渡して，環境を変更できることを意味します。**e** がない場合は，子プロセスは親プロセスの環境を受け継

ぎます。

spawn...ファミリィの各関数は，引数の指定に関するサフィックス(**l** または **v**)のいずれかを含んでいなければなりません。パス探索と環境の受け継ぎに関するサフィックス（**p** と **e**）はオプションです。たとえば次のようになります。

- **spawnl** は，独立した引数をとり，ルートおよびカレントディレクトリの中でだけ子プロセスを探し，親プロセスの環境を子プロセスに渡す。
- **spawnvpe** は，引数ポインタの配列をとり，子プロセスを探すのに環境変数 PATH を使い，子プロセスの環境を変更するために *envp* 引数を受け入れる。

spawn...関数は，少なくとも1つの引数（*arg0* または *argv*[0]）を子プロセスに渡さなくてはなりません。慣例上この引数は *path* です(違う値を使ってもエラーにはなりません)。
DOS 3.0以降では子プロセスで *path* を使用できますが，それより前のバージョンでは子プロセスは0番目の引数（*arg0* または *arg*[0]）の値を使うことはできません。
サフィックス **l** がついている場合，*arg0* は *path* を指し，*arg1, ..., argn* は新しい引数リストを構成する文字列を指します。*argn* の後ろの NULL はリストの最後であることを示します。
サフィックス **e** がついている場合は，新しい環境設定のリストを引数 *envp* を通して子プロセスに渡すことができます。*envp* は，char ポインタの配列で，次のような形式のヌル文字で終わる文字列を指しています。

```
envvar = value
```

envvar に *value* が代入されます。*envp*[]の最後の要素は NULL です。*envp*[0]が NULL のとき，子プロセスは親プロセスの環境設定を受け継ぎます。
arg0＋*arg1*＋...＋*argn*（または *argv*[0]＋*argv*[1]＋...＋*argv*[*n*]）の合計の長さは，引数を区切る空白文字も含めて，127バイト以下でなければなりません。ヌル文字はこれには入りません。
spawn...関数が呼び出されたときにオープンされているファイルは，子プ

ロセスにおいてもオープンされたままです。

戻り値

成功した場合，戻り値は子プロセスの終了ステータス（正常終了の場合は0）となります。子プロセスで**exit**を0以外の引数で呼び出せば，終了ステータスを0以外の値にセットすることができます。
エラーの場合は，**spawn...**関数は-1を返し，***errno***に次のいずれかの値をセットします。

E2BIG	引数リストが長すぎる
EINVAL	引数が正しくない
ENOENT	パス名またはファイル名が見つからない
ENOEXEC	exec フォーマットエラー
ENOMEM	メモリが不足した

関連項目

abort, atexit, _exit, exit, exec..., _fpreset, searchpath, system

例

```
/* SPAWNFAM.C: これを実行する際には，このあとの CHILD.C を
               .EXEファイルにコンパイルしておいてください．*/

#include <stdio.h>
#include <process.h>

status(int val)
{
   if (val == -1)
      printf("failed to start child process¥n");
   else
      if (val > 0) printf("child terminated abnormally¥n");
}

/* 次ページにつづく */
```

```
main()
{       /* 注意：以下の環境文字列は
           実行するマシンに合わせて変更してください．*/

   /* 環境文字列を作成する */
   char *envp[] =  { "PATH=A:¥¥",
                      "DUMMY=YES",
                    };

   /* pathname を作成する */
   char *pathname = "C:¥¥CHILDREN¥¥CHILD.EXE";

   /* 引数文字列を作成する */
   char *args[] =  { "CHILD.EXE", "1st", "2nd", NULL };

   printf("SPAWNL:¥n");
   status(spawnl(P_WAIT, pathname,args[0], args[1], NULL));

   printf("¥nSPAWNV:¥n");
   status(spawnv(P_WAIT, pathname,args));

   printf("¥nSPAWNLE:¥n");
   status(spawnle(P_WAIT, pathname,args[0], args[1], NULL, envp));

   printf("¥nSPAWNVPE:¥n");
   status(spawnvpe(P_WAIT, pathname,args, envp));
}


/* CHILD.C --- SPAWNFAM.C のための子プロセス */

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[])
{
   int i;
   char *path, *dummy;

   path = getenv("PATH");
   dummy = getenv("DUMMY");
   for (i = 0; i < argc; i++)
      printf("argv[%d] %s¥n", i, argv[i]);
   if (path)
      printf("PATH = %s¥n", path);
   if (dummy)
      printf("DUMMY = %s¥n", dummy);

   exit(0);  /* 親プロセスにエラーコード0を返す */
}
```

sprintf

機能　文字列に書式つき出力を書き込みます。

形式　int sprintf(char * *buffer*, const char * *format* [, *argument*,...]) ;

プロトタイプ　stdio.h

解説　**sprintf** は，*format* によって指される書式文字列中の書式指定を，*format* の後に続く各引数に適用し，書式化されたデータを文字列 * *buffer* に出力します。

sprintf は，最初の書式指定を最初の引数に，2つめの書式指定を2つめの引数に，というようにそれぞれ順番に適用していきます。書式指定は，後に続く引数と同じ数だけなければなりません。

* *buffer* にその文字列を格納する充分な領域があるかどうか確認するのはプログラマの責任です。

書式指定を含む詳細な情報については **printf** の解説を参照してください。

戻り値　**sprintf** は，出力したバイト数を返します。このバイト数には，最後のヌルバイトは含まれません。エラーの場合は EOF を返します。

可搬性　**sprintf** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**fprintf**, **printf**

例　**printf** を参照してください。

sqrt

機能 正の平方根を計算します。

形式 #include <math.h>

double sqrt (double *x*);

プロトタイプ math.h

解説 **sqrt** は，入力値の正の平方根を計算します。

戻り値 成功した場合，**sqrt** は，計算結果（*x* の正の平方根）を返します。

x が負の値であった場合には0を返し，***errno*** に次の値をセットします。

EDOM　　定義域エラー

可搬性 **sqrt** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目 **exp**, **log**, **pow**

srand

機能　乱数発生ルーチンを初期化します。

形式　void srand(unsigned *seed*)；

プロトタイプ　stdlib.h

解説　乱数発生ルーチンは，引数を1として **srand** を呼び出すことで初期化することができます。*seed* に適当な値を指定して **srand** を呼び出せば，乱数発生の新たな開始点をセットすることができます。

戻り値　ありません。

可搬性　**srand** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**rand**, **random**, **randomize**

sscanf

機能　文字列をスキャンして書式つき入力を行ないます。

形式　int sscanf(const char * *buffer*, const char * *format address*,...]) ;

プロトタイプ　stdio.h

解説　**sscanf** は，一連の入力フィールドをスキャンして，一度に1文字ずつ文字列から文字を読み込みます。次に，引数 *format* によって指される書式文字列中の書式指定にしたがって，各フィールドを書式化します。最後に，*format* の後に続く各引数が示しているアドレスに，書式化した入力を格納していきます。書式文字列中の書式指定の個数は，その後に続くアドレスの数と同じでなければなりません。

書式指定を含む詳細な情報については scanf の解説を参照してください。

sscanf は，いくつかの理由から，通常のフィールド終了文字（ホワイトスペース）に達する前に，特定のフィールドのスキャンをやめたり，あるいは読み込み全体をやめてしまうことがあります。こうした問題については，**scanf** の解説を参照してください。

戻り値　**sscanf** は，正しくスキャンし，変換し，格納した入力フィールドの数を返します。戻り値には，値を格納しなかった入力フィールドの数は含まれません。

文字列の終わりを読み込んだ場合は，戻り値は EOF になります。

値を格納したフィールドがなかった場合は，戻り値は0となります。

可搬性　**sscanf** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

関連項目　**fscanf**, **scanf**

stat

機能　オープンファイルに関する情報を得ます。

形式
```
#include <sys¥stat.h>
int stat(char * path, struct stat * statbuf);
```

プロトタイプ　sys¥stat.h

解説　**stat** は，*path* に指定されたオープンファイルまたはディレクトリに関する情報を **stat** 構造体の中に格納します。どちらの関数においても，*statbuf* は，**stat** 構造体(sys¥stat.h の中で定義されている)を指しています。**stat** 構造体には次のようなフィールドが含まれています。

st_mode	ファイルのモードに関する情報を与えるビットマスク
st_dev	そのファイルが置かれているディスクのドライブ番号
st_rdev	*st_dev* と同じ
st_nlink	整定数1にセット
st_size	オープンファイルの大きさ（バイト数）
st_atime	オープンファイルの一番最近修正された日時
st_mtime	*st_atime* と同じ
st_ctime	*st_atime* と同じ

stat 構造体には，これ以外にフィールドが3つありますが，DOS では意味を持たないので省略します。ビットマスクはオープンファイルのモードに関する情報を含んでおり，各ビットは次のような意味を持っています。

次のビットのうち，いずれか1つがセットされます。

S_IFREG	*path* が通常ファイルを指定している場合
S_IFDIR	*path* がディレクトリを指定している場合

次の一方または両方のビットがセットされます。

S_IWRITE　ユーザが書き出し許可を得ている場合
S_IREAD　ユーザが読み込み許可を得ている場合

ビットマスクにはユーザ実行ビットも含まれています。これは，オープンファイルの拡張子にしたがってセットされます。ビットマスクには，さらに読み込み/書き出しビットも含まれています。これらはファイルの許可モードにしたがってセットされます。

戻り値　**stat** は，そのオープンファイルに関する情報をうまく得られた場合は0を返します。エラーの場合は-1を返し，***errno*** に次の値をセットします。

ENOENT　パス名またはファイル名が見つからない

関連項目　**access, chmod, fstat, stat**

_status87

機能　浮動小数点ステータスを得ます。

形式　unsigned int _status87(void);

プロトタイプ　float.h

解説　**_status87** は，浮動小数点ステータスワードを取り出します。浮動小数点ステータスワードは，8087/80287のステータスワードと8087/80287例外ハンドラによって検出される他の条件を組み合わせたものです。

戻り値　**戻り値**の各ビットが浮動小数点ステータスを示します。**_status87** が返すビットの意味については float.h を参照してください。

関連項目　**_clear87**, **_control87**, **_fpreset**

例　**_control87** を参照してください。

stime

機能　時刻をセットする

形式

```
# include <time.h>
int stime(time_t * tp);
```

プロトタイプ　time.h

解説　**stime** は，システムの時刻と日付をセットします。*tp* は，GMT1970年1月1日の00：00：00から，設定する日付時刻までの経過秒数を表わす値を指します。

戻り値　**stime** は，成功した場合は0を返します。

可搬性　**stime** は UNIX システムで使用できます。

関連項目　**asctime, ftime, gettime, gmtime, localtime, time, tzset**

stpcpy

機能　ある文字列を他の文字列にコピーします。

形式　char * stpcpy(char * *dest*, const char * *src*)；

プロトタイプ　string.h

解説　**stpcpy** は，*src* から *dest* へコピーを行ないます。ヌル終了文字に出会ったところでコピーをやめます。

戻り値　**stpcpy** は，*dest* + **strlen**(*src*)を返します。

可搬性　**stpcpy** は UNIX システムで使用できます。

strcat

機能　ある文字列を他の文字列に付け加えます。

形式　char * strcat(char * *dest*, const char * *src*)；

プロトタイプ　string.h

解説　**strcat** は，*src* のコピーを *dest* の最後に付け加えます。得られる文字列の長さは，**strlen**(*dest*) + **strlen**(*src*)になります。

戻り値　**strcat** は，連結された文字列へのポインタを返します。

可搬性　**strcat** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

strchr

機能　文字列を調べて，指定の文字が最初に現われる位置を探します。

形式　char ＊ strchr(const char ＊ *s*, int *c*)；

プロトタイプ　string.h

解説　**strchr** は，文字列を先頭から調べて，指定の文字を探します。**strchr** は，文字列 *s* の中で最初に現われる文字 *c* を見つけます。
ヌル文字も文字列の一部と考えるので，たとえば次の呼び出しは，

```
strchr(strs, 0)
```

文字列 *s* の最後のヌル文字を指すポインタを返します。

戻り値　**strchr** は，*s* の中で最初に見つかった文字 *c* へのポインタを返します。

可搬性　**strcat** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

strcmp

機能　ある文字列と他の文字列を比較します。

形式　int strcmp(const char * *s1*, const char * *s2*)；

プロトタイプ　string.h

解説　**strcmp** は，*s1* と *s2* 中の各文字を unsigned char として比較します。比較は各文字列の最初の文字から始め，異なる文字が見つかるまで，あるいは文字列の終わりに達するまで続けます。

戻り値　**strcmp** は次のような値を返します。

<0	*s1* が *s2* より小さい
==0	*s1* と *s2* が等しい
>0	*s1* が *s2* より大きい

可搬性　**strcmp** は UNIX システムで使用でき，ANSI C と互換性があります。

strcmpi

機能 文字ケースを区別せずに，2つの文字列を比較します。

形式 # include <string.h>
int strcmpi(const char * *s1*, const char * *s2*);

プロトタイプ string.h

解説 **strcmpi** は，文字ケース（大文字/小文字）を区別せずに，*s1* と *s2* 中の各文字を unsigned char として比較します。比較は各文字列の最初の文字から始め，異なる文字が見つかるまで，あるいは文字列の終わりに達するまで続けます（**stricmp** と同じですが，**strcmpi** はマクロです）。

strcmpi は，*s1*（またはその一部）を *s2*（またはその一部）と比較した結果に基づいて負，ゼロ，正の値を返します。

strcmpi は **stricmp** と同じです。**strcmpi** は，string.h の中で **stricmp** の呼び出しに変換されるマクロとして定義されています。したがって，**strcmpi** を使うときにはヘッダファイル string.h をインクルードしなければなりません。このマクロは他のCコンパイラとの互換性のために用意されています。

戻り値 **strcmpi** は次のような値を返します。

<0	*s1* が *s2* より小さい
＝＝0	*s1* と *s2* が等しい
>0	*s1* が *s2* より大きい

これら6つの関数は符号を考慮した比較を行ないます。

strcpy

機能　ある文字列を他の文字列にコピーします。

形式　char * strcpy(char * *dest*, const char * *src*);

プロトタイプ　string.h

解説　**strcpy** は，文字列 *src* を *dest* にコピーします。最後のヌル文字をコピーしたところでコピーが終わります。

戻り値　**strcpy** は，*dest* を返します。

可搬性　**strcpy** は UNIX システムで使用でき，ANSI C と互換性があります。

strcspn

機能　文字列をスキャンし，指定の文字集合に含まれない文字のみからなる最初の部分を探します。

形式　# include <string.h>

```
size_t strcspn(const char * s1, const char * s2) ;
```

プロトタイプ　string.h

解説　**strcspn** は，文字列 *s1* を調べて，文字列 *s2* に含まれていない文字のみからなる最初の部分を見つけます。

戻り値　**strcspn** は，*s1* 中の，*s2* に含まれていない文字のみからなる最初の部分の長さを返します。

可搬性　**strcspn** は UNIX システムで使用でき，ANSI C と互換性があります。

strdup

機能　文字列を新たに作成した記憶場所にコピーします。

形式　char ＊ strdup(const char ＊ *s*)；

プロトタイプ　string.h

解説　**strdup** は，文字列 s の複製を作ります。複製を格納する領域は，**malloc** を呼び出して確保します。領域の大きさは(**strlen**(*s*)＋1)バイトです。

戻り値　**strdup** は，複製された文字列が格納されている記憶場所を指すポインタを返します。記憶場所が確保できなかった場合は NULL を返します。

可搬性　**strdup** は UNIX システムで使用できます。

関連項目　**free**

_strerror

機能 エラーメッセージ文字列を指すポインタを返します。

形式 char *_strerror(const char *s);

プロトタイプ string.h, stdio.h

解説 **_strerror** は，ユーザ専用のエラーメッセージを生成するための関数です。これは，ヌルで終わるエラーメッセージ文字列を指すポインタを返します。

- *s* が NULL の場合，戻り値には一番新しいシステムエラーメッセージが含まれています。この文字列はヌル文字で終わっています。
- *s* が NULL でない場合は，戻り値には *s*（ユーザ定義のメッセージ），コロン，空白，一番新しいシステムエラーメッセージ，改行文字が含まれています。*s* の長さは94文字以下でなければなりません。

Turbo C のバージョン1.0では，**_strerror** は **strerror** と同じものでした。

戻り値 **_strerror** は，エラーメッセージ文字列を指すポインタを返します。この文字列は静的なバッファに格納されるので，**_strerror** を呼び出すたびに上書きされます。

関連項目 **perror**, **strerror**

strerror

機能 エラーメッセージ文字列を指すポインタを返します。

形式 char * strerror(int *errnum*)；

プロトタイプ stdio.h, string.h

解説 **strerror** は，エラー番号 *errnum* を引数にとり，これに対応するエラーメッセージ文字列を指すポインタを返します。

戻り値 **strerror** は，エラーメッセージ文字列を指すポインタを返します。この文字列は静的なバッファに格納されるので，**strerror** を呼び出すたびに上書きされます。

可搬性 **strerror** は ANSI C と互換性があります。

関連項目 **perror**, **_strerror**

stricmp

機能　大文字小文字を区別せずに，2つの文字列を比較します。

形式　int　stricmp(const char * *s1*, const char * *s2*)；

プロトタイプ　string.h

解説　**stricmp** は，文字ケース（大文字/小文字）を区別せずに，*s1* と *s2* 中の各文字を unsigned char として比較します。比較は各文字列の最初の文字から始め，異なる文字が見つかるまで，あるいは文字列の終わりに達するまで続けます。

stricmp は，*s1*（またはその一部）を *s2*（またはその一部）と比較した結果に基づいて負，ゼロ，正の値を返します。

stricmp は **strcmpi** と同じですが，**strcmpi** は，string.h の中で **stricmp** の呼び出しに変換されるマクロとして定義されています。

戻り値　**stricmp** は次のような値を返します。

<0	*s1* が *s2* より小さい
==0	*s1* と *s2* が等しい
>0	*s1* が *s2* より大きい

strlen

機能　文字列の長さを計算します。

形式　# include <string.h>
size_t strlen(const char * *s*);

プロトタイプ　string.h

解説　**strlen** は，文字列 *s* の長さを計算します。

戻り値　**strlen** は，*s* 中の文字の個数を返します。ヌル文字はカウントしません。

可搬性　**strlen** は UNIX システムで使用でき，ANSI C と互換性があります。

strlwr

機能　文字列の中の大文字を小文字に変換します。

形式　char * strlwr(char * *s*);

プロトタイプ　string.h

解説　**strlwr** は，文字列 *s* の中の大文字（A～Z）を小文字（a～z）に変換します。英字以外の文字は変更されません。

戻り値　**strlwr** は，文字列 *s* へのポインタを返します。

関連項目　**strupr**

strncat

機能 ある文字列の一部分を他の文字列に付け加えます。

形式 # include <string.h>

char * strncat(char * *dest*, const char * *src*, size_t *maxlen*);

プロトタイプ string.h

解説 **strncat** は，*src* の最大 *maxlen* 個の文字を，*dest* の終わりに付け加え，最後にヌル文字を付加します。得られる文字列の長さの上限は，**strlen**(*dest*)＋*maxlen* になります。

戻り値 **strncat** は，*dest* を返します。

可搬性 **strncat** は UNIX システムで使用でき，ANSI C と互換性があります。

strncmp

機能 ある文字列の一部分と他の文字列と比較します。

形式 # include <string.h>

int strncmp(const char * *s1*, const char * *s2*, size_t *maxlen*) ;

プロトタイプ string.h

解説 **strncmp** は，**strcmp** と同様に，文字列 *s1* と *s2* 中の各文字を unsigned char として比較します。比較は各文字列の最初の文字から始め，異なる文字が見つかるまで，または *maxlen* 個の文字を比較するまで，あるいは文字列の終わりに達するまで続けます。

strncmp は，*s1*（またはその一部）を *s2*（またはその一部）と比較した結果に基づいて負，ゼロ，正の値を返します。

戻り値 **strncmp** は次のような値を返します。

<0	*s1* が *s2* より小さい
==0	*s1* と *s2* が等しい
>0	*s1* が *s2* より大きい

可搬性 **strncmp** は UNIX システムで使用でき，ANSI C と互換性があります。

strncmpi

機能 ある文字列の一部分と他の文字列の一部分を，大文字小文字を区別せずに比較します。

形式 # include <string.h>

int strncmpi(const char * *s1*, const char * *s2*, size_t *n*)；

プロトタイプ string.h

解説 **strncmpi** は，文字ケース（大文字/小文字）を区別せずに，*s1* と *s2* 中の最大 *n* バイトの文字を unsigned char として比較します。比較は各文字列の最初の文字から始め，異なる文字が見つかるまで，または *n* 個の文字を比較するまで，あるいは文字列の終わりに達するまで続けます（**strncmpi** は **strnicmp** と同じですが，**strncmpi** はマクロです)。

strncmpi は，*s1*（またはその一部）を *s2*（またはその一部）と比較した結果に基づいて負，ゼロ，正の値を返します。

strncmpi は **strnicmp** と同じですが，**strncmpi** は，string.h の中で **strnicmp** の呼び出しに変換されるマクロとして定義されています。したがって，**strncmpi** を使うときには，ヘッダファイル string.h をインクルードしなければなりません。

戻り値 **strncmpi** は次のような値を返します。

<0	*s1* が *s2* より小さい場合
==0	*s1* と *s2* が等しい場合
>0	*s1* が *s2* より大きい場合

strncpy

機能 ある文字列の一部を他の文字列にコピーします。必要な場合には削除あるいはパディングを行ないます。

形式 # include <stdio.h>

char ＊ strncpy(char ＊ *dest*, const char ＊ *src*, size_t *maxlen*)；

プロトタイプ string.h

解説 **strncpy** は，最大 *maxlen* 個の文字を *src* から *dest* にコピーします。長すぎれば削り，短かすぎればヌル文字を *dest* に埋めこみます。*dest* には，*src* の長さが *maxlen* 以上の場合は，最後にヌル文字はつけられません。

戻り値 **strncpy** は，*dest* を返します。

可搬性 **strncpy** は UNIX システムで使用でき，ANSI C と互換性があります。

strnicmp

機能　ある文字列の一部分と他の文字列の一部分を，大文字小文字を区別せずに比較します。

形式　# include <string.h>
int strnicmp(const char * *s1*, const char * *s2*, size_t *maxlen*)；

プロトタイプ　string.h

解説　**strnicmp** は，文字ケース（大文字/小文字）を区別せずに，*s1* と *s2* 中の最大 n バイトの文字を unsigned char として比較します。比較は各文字列の最初の文字から始め，異なる文字が見つかるまで，または *n* 個の文字を比較するまで，あるいは文字列の終わりに達するまで続けます(**strnicmp** は **strncmpi** と同じですが，**strncmpi** はマクロです)。
strnicmp は，*s1*（またはその一部）を *s2*（またはその一部）と比較した結果に基づいて負，ゼロ，正の値を返します。
strncmpi は **strnicmp** と同じですが，**strncmpi** は，string.h の中で **strnicmp** の呼び出しに変換されるマクロとして定義されています。

戻り値　**strnicmp** は次のような値を返します。

<0	*s1* が *s2* より小さい場合
＝＝0	*s1* と *s2* が等しい場合
>0	*s1* が *s2* より大きい場合

strnset

機能　文字列中の指定個数の文字に指定の文字をセットします。

形式　# include <string.h>

char ＊ strnset(char ＊ *s*, int *ch*, size_t *n*)；

プロトタイプ　string.h

解説　**strnset** は，文字列 *s* の先頭から *n* バイトに文字 *ch* をコピーします。

n>**strlen**(*s*)の場合，*n* は **strlen**(*s*)に置き換えられます。

コピーは，*n* 文字をセットするまで，あるいはヌル文字が見つかるまで行なわれます。

戻り値　**strnset** は *s* を返します。

strpbrk

機能 文字列を調べて，指定した文字集合中の文字が最初に現われる場所を探します。

形式 char * strpbrk(const char * *s1*, const char * *s2*)；

プロトタイプ string.h

解説 **strpbrk** は，文字列 *s1* を調べて，文字列 *s2* に含まれるどれか1文字が最初に現われる位置を探します。

戻り値 **strpbrk** は，*s2* 内の文字のどれかが最初に現われる場所を指すポインタを返します。見つからない場合は NULL を返します。

可搬性 **strpbrk** は UNIX システムで使用でき，ANSI C と互換性があります。

strrchr

機能　　文字列を調べて，指定した文字が最後に現われる場所を探します。

形式　　char ＊ strrchr(const char ＊ *s*, int *c*)；

プロトタイプ　string.h

解説　　**strrchr** は，文字列を終わりから逆方向に調べて，指定の文字を探します。**strrchr** は，文字列 *s* の中で最後に現われる文字 *c* を見つけます。ヌル文字も文字列の一部と考えます。

戻り値　　**strrchr** は，文字 *c* が最後に現われる場所を指すポインタを返します。見つからない場合は NULL を返します。

可搬性　　**strrchr** は UNIX システムで使用でき，ANSI C と互換性があります。

strrev

機能　　文字列を逆転させます。

形式　　char ＊ strrev(char ＊ *s*)；

プロトタイプ　string.h

解説　　**strrev** は，文字列 *s* の中のすべての文字を，最後のヌル文字は除いて逆順に並べ換えます（たとえば，string¥0は gnirts¥0になります)。

戻り値　　**strrev** は，逆順になった文字列を指すポインタを返します。エラーの戻り値はありません。

strset

機能　文字列中のすべての文字を指定の文字に変更します。

形式　char * strset(char * *s*, int *ch*)；

プロトタイプ　string.h

解説　**strset** は，文字列 *s* の中のすべての文字に文字 *ch* をセットします。ヌル文字に出会ったところで終了します。

戻り値　**strset** は，*s* を返します。

関連項目　**setmem**

strspn

機能　文字列を調べて，指定の文字集合に含まれる文字のみからなる最初の部分を探します。

形式　# include <string.h>
size_t strspn(const char * *s1*, const char * *s2*)；

プロトタイプ　string.h

解説　**strspn** は，文字列 *s1* を調べて，文字列 *s2* に含まれる文字のみからなる最初の部分を探します。

戻り値　**strspn** は，*s1* 中の，*s2* に含まれる文字のみからなる最初の部分の長さを返します。

可搬性　**strspn** は UNIX システムで使用でき，ANSI C と互換性があります。

strstr

機能　文字列を調べて，指定の部分文字列が現われる場所を探します。

形式　char ＊ strstr(const char ＊ *s1*, const char ＊ *s2*)；

プロトタイプ　string.h

解説　**strstr** は，文字列 *s1* を調べて，部分文字列 *s2* が最初に現われる位置を探します。

戻り値　**strstr** は，*s2* を含む *s1* 中の要素を指す（*s1* 中の *s2* を指す）ポインタを返します。*s2* が *s1* 中に存在しない場合は NULL を返します。

可搬性　**strstr** は UNIX システムで使用でき，ANSI C と互換性があります。

strtod

機能　文字列を double の値へ変換します。

形式

```
# include <stdlib.h>
double strtod(const char * s, char ** endptr) ;
```

プロトタイプ　stdlib.h

解説

strtod は，文字列 *s* を double の値に変換します。*s* が double の値として解釈されるためには，次のような形式になっていなければなりません。

```
[ws][sn][ddd][.][ddd][fmt[sn]ddd]
```

各部分は次のようになります。

[*ws*]	ホワイトスペース（なくてもよい）
[*sn*]	符号（＋または-，なくてもよい）
[*ddd*]	数字の並び（なくてもよい）
[*fmt*]	e か E（なくてもよい）
[.]	小数点（なくてもよい）

strtod はさらに，＋INF および-INF を，プラス無限大およびマイナス無限大として認識します。また，＋NAN および-NAN を非数（Not a Number）として解釈します。

strtod で double に変換できる文字列は次のようなものです。

```
+1231.1981 e-1
502.85E2
-2010.952
```

strtod は，double の数値として解釈できない文字があると読み込みをやめます。*endptr* が NULL でなければ，*endptr* を，読み込みをやめた文字を指すようにセットします(＊ *endptr* ＝ *&stopper*)。

戻り値　**strtod** は，*s* を変換した値を double として返します。オーバーフローが起きた場合は HUGE_VAL を返します。

可搬性　**strtod** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**atof**

strtok

機能　文字列からトークンを次々に切り出します。トークンは2番目に指定した文字列の中に入っている文字によって区切られます。

形式　char * strtok(char * *s1*, const char * *s2*);

プロトタイプ　string.h

解説　**strtok** は，文字列 *s1* は，*s2* に含まれている文字（区切り子）によって区切られるトークンをいくつか含んでいるものと考えます。

strtok を最初に呼び出すと，*s1* 中の最初のトークンの先頭文字を指すポインタを返し，返されたトークンの直後にヌル文字を書きます。その後，最初の引数として NULL を指定して **strtok** を連続して呼び出すと，**strtok** は，*s1* にトークンがなくなるまで次のトークンを探していきます。

区切り子文字列 *s2* は，呼び出しごとに変化してもかまいません。

戻り値　**strtok** は，*s1* 中の見つかったトークンへのポインタを返します。それ以上トークンがない場合には NULL を返します。

可搬性　**strtok** は UNIX システムで使用でき，ANSI C と互換性があります。

例

```
/*  strtok - この例題は，strtokを使って日付を表す文字列を
    分割します。形式の異なる日付（12/3/87 ; Dec.12,1987 ;
    January 15, 1987 ; 12-FEB-87, etc.）を扱う場合，区切り
    文字の文字列には，必要な文字（ピリオド，空白，カンマ，
    マイナスなど）を入れる必要があります。
    出力には区切り文字は含まれません。
*/
#include <stdio.h>
#include <string.h>

main ()
{
   char *ptr;

   ptr = strtok ("FEB.14,1988", ". ,-/" );
   while (ptr !=NULL)
   {
      printf ("ptr = %s¥n", ptr);
      ptr = strtok (NULL, ". ,-/" );
   }
}
```

プログラム出力

```
ptr = FEB
ptr = 14
ptr = 1988
```

strtol

機能　文字列を long の値に変換します。

形式　long strtol(const char ＊ *s*, char ＊＊ *endptr*, int *radix*)；

プロトタイプ　stdlib.h

解説　**strtol** は，文字列 *s* を long の値に変換します。*s* が long の値として解釈されるためには，次のような形式になっていなければなりません。

```
[ws][sn][0][x][ddd]
```

各部分は次のようになります。

[*ws*]	ホワイトスペース（なくてもよい）
[*sn*]	符号（＋または－，なくてもよい）
[0]	ゼロ（なくてもよい）
[*x*]	x または X（なくてもよい）
[*ddd*]	数字（なくてもよい）

strtol は，認識できない文字のところで文字列を読むのをやめます。

radix が2～36の場合は，long 型整数は *radix* を基底として表現されます。

radix が0の場合は，*s* の先頭の何文字かによって何進法であるかが決まります。

最初の文字	2番目の文字	文字列
0	1～7	8進表記
0	x または X	16進表記
1～9		10進表記

radix が1の場合は，無効な値とみなされます。*radix* が負あるいは36を越える場合は，エラーとなります。

radix として正しくない値を指定すると，結果は0となり，次の文字ポインタを開始文字列ポインタにセットします。

s が8進数として解釈される場合は，0～7以外の文字は認識されません。

s が10進数として解釈される場合は，0～9以外の文字は認識されません。

s の値が他の数を基底とする数として解釈される場合は，数字と，その基底で使用される英字だけが認識されます。たとえば，基底が20のときは，0～9およびA～Jが認識されます。

戻り値　**strtol** は，文字列を変換した値を返します。エラーの場合は0を返します。

可搬性　**strtol** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**atoi, atol, strtoul**

strtoul

機能　文字列を unsigned long の値に変換します。

形式　unsigned long strtoul(const char * *s*, char * * *endptr*, int *radix*)；

プロトタイプ　stdlib.h

解説　**strtoul** は，文字列 *s* を unsigned long 型の値へ変換する以外は，**strtol** と同じです。詳細については **strtol** を参照してください。

戻り値　**strtoul** は，変換した値を unsigned long で返します。エラーの場合は0を返します。

可搬性　**strtoul** は ANSI C と互換性があります。

関連項目　**atol**, **strtol**

strupr

機能　文字列中の小文字を大文字に変換します。

形式　char ＊ strupr(char ＊ *s*)；

プロトタイプ　string.h

解説　**strupr** は，文字列 *s* の中の小文字（a～z）を大文字（A～Z）に変換します。英字以外の文字は変更されません。

戻り値　**strupr** は，*s* を返します。

関連項目　**strlwr**

swab

機能　バイトを入れ替えます。

形式　void swab(char * *from*, char * *to*, int *nbytes*)；

プロトタイプ　stdlib.h

解説　**swab**は，文字列*from*から文字列*to*へ，*nbytes*バイトをコピーします。このとき，隣り合う偶数位置と奇数位置のバイトが入れ替えられます。これはバイトの並べ方が異なるマシンからのデータを処理する場合に便利です。*nbytes*は偶数でなければなりません。

戻り値　ありません。

可搬性　**swab**はUNIXシステムで使用できます。

system

機能　DOS コマンドを実行します。

形式　int system(const char * *command*) ;

プロトタイプ　stdlib.h, process.h

解説　**system** は, C プログラムの中から, DOS の COMMAND.COM を呼び出して, 文字列 *command* で与えられた DOS コマンド, バッチファイルあるいは他のプログラムを実行します。
実行されるプログラムは, カレントディレクトリか, 環境変数 PATH にセットされているディレクトリに置かれていなければなりません。
COMMAND.COM ファイルを探すために, 環境変数 COMSPEC が使われます。したがって, COMMAND.COM はカレントディレクトリに存在する必要はありません。

戻り値　**system** は, 成功した場合は0, 失敗した場合には-1を返します。

可搬性　**system** は UNIX システムで使用でき, ANSI C と互換性があります。K&R でも定義されています。

関連項目　**exec..., _fpreset, searchpath, spawn...**

tan

機能　正接を求めます。

形式　# include <math.h>
double tan(double *x*);

プロトタイプ　math.h

解説　**tan** は，入力値の正接（タンジェント）を計算します。角度 *x* はラジアン単位で指定します。
tan のエラー処理は，関数 **matherr** を使って変更することができます。

戻り値　**tan** は，*x* の正接を返します。すべての有効な角度に対して値を返しますが，$-\pi/2$ や $\pi/2$ に近い角度の場合は0を返し，***errno*** に次の値をセットします。

ERANGE　　値域エラー

可搬性　**tan** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**acos, asin, atan, atan2, cos, cosh, sin, sinh, tanh**

tanh

機能　双曲線正接を求めます。

形式　# include <math.h>
double tanh(double *x*);

プロトタイプ　math.h

解説　**tanh** は，実数引数の双曲線正接（ハイパボリックタンジェント）を計算します。
tanh のエラー処理は，**matherr** 関数を通して変更することができます。

戻り値　**tanh** は，*x* の双曲線正接を返します。

可搬性　**tanh** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**acos, asin, atan, atan2, cos, cosh, sin, sinh, tan**

tell

機能　ファイルポインタの現在位置を得ます。

形式　long tell(int *handle*)；

プロトタイプ　io.h

解説　**tell** は，*handle* に結びつけられているファイルポインタの現在位置を得て，それをファイルの先頭からのバイト数として示します。

戻り値　**tell** は，ファイルポインタの現在位置を返します。エラーの場合は-1L を返し，***errno*** に次の値をセットします。

EBADF　　ファイル番号が正しくない

可搬性　**tell** はすべての UNIX システムで使用できます。

関連項目　**fgetpos**，**fseek**，**ftell**，**lseek**

time

機能　時刻を得ます。

形式　# include <time.h>

time _t time(time _t ※ *timer*)

プロトタイプ　time.h

解説　**time** は，GMT（グリニッジ標準時刻）1970年1月1日の00：00：00から現在までの経過時間を秒単位で与え，その値を *timer* が指す記憶場所に格納します。*timer* には，NULL でないポインタを与える必要があります。

戻り値　**time** は，上に述べた時刻を秒単位で返します。

可搬性　**time** は UNIX システムで使用でき，ANSI C と互換性があります。

関連項目　**asctime, ctime, difftime, ftime, gettime, gmtime, localtime, settime, stime, tzset**

tmpfile

機能 “スクラッチ”ファイルをバイナリモードでオープンします。

形式

```
#include <stdio.h>
FILE * tmpfile(void);
```

プロトタイプ stdio.h

解説 **tmpfile** は，テンポラリ（一時的）バイナリファイルを作成し，更新用（”*w*+*b*”）としてオープンします。このファイルは，クローズしたりプログラムが終了した時点で，自動的に削除されます。

戻り値 **tmpfile** は，作成したテンポラリファイルのストリームを指すポインタを返します。ファイルを作成できなかった場合は NULL を返します。

可搬性 **tmpfile** は UNIX システムで使用でき，ANSI C と互換性があります。

tmpnam

機能　ユニークなファイル名を作ります。

形式　char ＊ tmpnam(char ＊ *s*)；

プロトタイプ　stdio.h

解説　**tmpnam** はユニークなファイル名を作り出します。そのファイル名はテンポラリ（一時的）ファイルの名前として安全に使うことができます。**tmpnam** は，最大 TMP_MAX 回まで，呼び出すたびに異なる文字列を発生します。TMP_MAX は，stdio.h の中で65535と定義されています。**tmpnam** の引数 *s* は，NULL か，最低 *L_tmpnam* 個の文字を要素に持つ配列へのポインタです。*L_tmpnam* は stdio.h の中で定義されています。*s* が NULL の場合，**tmpnam** は作成したテンポラリファイル名を内部の静的オブジェクトに残し，それを指すポインタを返します。*s* が NULL でない場合は，テンポラリファイル名を *s* が指す配列の中に格納して *s* を返します。

注意：**tmpnam** を使ってテンポラリファイルを作成した場合，そのファイルを削除するのはプログラマの責任です（たとえば **remove** を呼び出す）。そのファイルは自動的には消されません。

戻り値　*s* が NULL の場合，**tmpnam** は内部の静的オブジェクトを指すポインタを返します。それ以外の場合は *s* を返します。

可搬性　**tmpnam** は UNIX システムで使用でき，ANSI C と互換性があります。

toascii

機能　整数を ASCII 文字に変換します。

形式　int toascii(int *c)*;

プロトタイプ　ctype.h

解説　**toascii** は，整数 *c* の下位7ビットをクリアして，ASCII 文字に変換するマクロです。返される値の範囲は0～127になります。

戻り値　**toascii** は，*c* を変換した値を返します。

可搬性　**toascii** は UNIX システムで使用できます。

_tolower

機能　　文字を小文字に変換します。

形式　　# include <ctype.h>
int _tolower(int *ch)*;

プロトタイプ　　ctype.h

解説　　**_tolower** は，整数 *ch*（範囲は EOF～255）を小文字（a～z）の値に変換するマクロです。これは，*ch* が大文字であるとわかっているときだけ使用すべきです。
_tolower は，**tolower** と同じ変換を行ないますが，**_tolower** はマクロなので，これを使うときには ctype.h をインクルードしなければなりません。

戻り値　　**_tolower** は，*ch* が大文字の場合には変換した値を返します。そうでない場合の結果は未定義です。

可搬性　　**_tolower** は UNIX システムで使用できます。

tolower

機能　　文字を小文字に変換します。

形式　　int tolower(int *ch)*;

プロトタイプ　　ctype.h

解説　　**tolower** は，整数 *ch*（範囲は EOF～255）を小文字（a～z）の値に変換する関数です。*ch* が大文字の場合にのみ変換が行なわれ，他の文字の場合はそのままです。

戻り値　　**tolower** は，*ch* が大文字の場合には変換した値を返します。そうでない場合には，*ch* をそのまま返します。

可搬性　　**tolower** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

_toupper

機能　文字を大文字に変換します。

形式
```
# include <ctype.h>
int _toupper(int ch);
```

プロトタイプ　ctype.h

解説　**_toupper** は，整数 *ch* を対応する大文字の値に変換するマクロです。これは，*ch* が小文字であるとわかっているときだけ使用すべきです。

_toupper は **toupper** と同じ変換を行ないますが，これはマクロなので，使うときには ctype.h をインクルードしなければなりません。

戻り値　**_toupper** は，*ch* が小文字の場合には変換した値を返します。そうでない場合の結果は未定義です。

可搬性　**_toupper** は UNIX システムで使用できます。

toupper

機能　　文字を大文字に変換します。

形式　　int toupper(int *ch)*；

プロトタイプ　ctype.h

解説　　**toupper** は，整数 *ch*（範囲は EOF～255）を大文字（A～Z）の値に変換する関数です。*ch* が小文字の場合にのみ変換が行なわれ，他の文字の場合はそのままです。

戻り値　　**toupper** は，**ch** が小文字の場合には変換した値を返します。そうでない場合には，**ch** をそのまま返します。

可搬性　　**toupper** は UNIX システムで使用でき，ANSI C と互換性があります。K&R でも定義されています。

tzset

機能 グローバル変数 ***daylight***, ***timezone***, ***tzname*** の値をセットします。

形式

```
# include <time.h>
void tzset(void);
```

プロトタイプ time.h

解説 **tzset** は，環境変数 TZ にしたがって，***daylight***, ***timezone***, ***tzname*** の3つのグローバル変数の値をセットします。ライブラリ関数 **ftime** と **localtime** は，地方時間帯がどこであっても正しいグリニッチ標準時（GMT）を得るために，この3つのグローバル変数を使用します。環境変数 TZ に設定される文字列の形式は以下の通りです。

```
TZ = zzz[+/-]d[d][lll]
```

zzz は現在の時間帯の名前を表わす3文字の文字列です。3文字すべてが必要です。たとえば，文字列"PST"は大平洋標準時（Pacific Standard Time）を表わすために使用されます。

[+/-]*d*[*d*]は，符号付きの1桁以上の数値を持つ必須のフィールドです(符号は正の値の場合には省略できます)。この数値は，その時間帯と GMT との差を示す時間数です。正の数の場合は GMT から西に，負の数の場合は GMT から東に向かって調整されます。たとえば，この数が5であれば EST，+8ならば PST，-1ならヨーロッパ大陸というようになります。この数値は，グローバル変数 ***timezone*** を計算する際に使われます。***timezone*** は，GMT と地方時間帯との差を秒単位で保持しています。

lll は，地方時間帯夏時間を表わす省略可能なフィールドです。たとえば" PDT"は，太平洋夏時間（Pacific Daylight Savings Time）を表わすために使われます。このフィールドが存在すれば，グローバル変数 ***daylight*** はゼロでない値にセットされることになります。

環境変数 TZ が存在しないか，存在しても上記のような形式ではない場合，グローバル変数 ***daylight***, ***timezone***, ***tzname*** へ代入するために，デフォ

ルトの TZ = "EST5EDT"が用いられます。

グローバル変数 ***tzname***[0]は，TZ 環境文字列から取り出した地方時間帯名の3文字の文字列を指しています。***tzname***[1]は，TZ 環境文字列から取り出した地方時間帯夏時間名を表わす3文字の文字列を指しています。夏時間帯名が無い場合には，***tzname***[1]はヌル文字列を指します。

戻り値　　ありません。

可搬性　　**tzset** は UNIX および XENIX システムで使用できます。

参照項目　　**asctime, ctime, ftime, gmtime, localtime, stime, time**

例

```
#include <stdlib.h>
#include <time.h>

main()
{
    time_t td;

    putenv("TZ=PST8PDT");        /* 太平洋夏時間 */
    tzset();
    time(&td);                   /* 現在の時刻/日付を得る */
    printf("Current time = %s¥n", asctime(localtime(&td)));
}
```

ultoa

機能　unsigned long を文字列に変換します。

形式　char ＊ ultoa(unsiged long *value*, char ＊ *string*, int *radix*)；

プロトタイプ　stdlib.h

解説　**ultoa** は，*value* の値をヌルで終わる文字列に変換し，結果を *string* に代入します。*value* は unsigned long 型です。
radix には，*value* を変換する際に使用する基底を指定します。この値は2～36の値でなければなりません。**ultoa** は，オーバーフローのチェックは行ないません。また，*value* が負で，*radix* が10であると，負符号(-)はセットされません。

注意：*string* 用に確保する領域は，返される文字列を格納するのに充分な大きさでなければなりません。**ultoa** が返すのは最大33文字です。

戻り値　**ultoa** は，*string* を返します。エラーの戻り値はありません。

関連項目　**itoa**, **ltoa**

ungetc

機能　入力ストリームへ1文字をプッシュバックします。

形式

```
#include <stdio.h>
int ungetc(int c, FILE * stream);
```

プロトタイプ　stdio.h

解説　**ungetc** は，入力ストリーム *stream* に1文字を返し（プッシュバック）ます。*stream* は入力用にオープンされたものでなければなりません。この文字は，次に *stream* に対する **getc** あるいは **fread** の呼び出しによって返されます。文字1個は無条件でプッシュバックすることができます。**getc** を呼び出さずに再び **ungetc** を呼び出すと，前にプッシュバックした文字を消してしまうことになります。**fflush**, **fseek**, **fsetpos**, あるいは **rewind** の呼び出しは，プッシュバックされた文字のすべてのメモリを消去します。

戻り値　**ungetc** は，成功した場合はプッシュバックした文字を返し，操作に失敗した場合は EOF を返します。

可搬性　**ungetc** は UNIX システムで使用でき，ANSI C と互換性があります

関連項目　**fgetc**，**getc**，**getchar**

ungetch

機能　キーボードバッファに1文字をプッシュバックします。

形式　int ungetch(int *ch*)；

プロトタイプ　conio.h

解説　**ungetch** は，コンソールに1文字をプッシュバックします。この文字は次の文字入力で読むことができます。**ungetch** は，読み込みをせずに2回以上続けて呼び出されるとエラーになります。

戻り値　**ungetch** は，成功した場合はプッシュバックした文字を返し，エラーの場合は EOF を返します。

可搬性　**ungetch** は UNIX システムで使用できます。

関連項目　**getch, getche**

unixtodos

機能　日付と時刻を DOS フォーマットに変換します。

形式　# include <dos.h>

void unixtodos(long *time*, struct date * *d*, struct time * *t*)；

プロトタイプ　dos.h

解説　**unixtodos** は，*time* の中に与えられている UNIX 形式の時刻を DOS 形式に変換し，*d* および *t* が指している **date** および **time** 構造体を埋めます。

戻り値　ありません。

可搬性　**unixtodos** は MS-DOS に特有の関数です。

関連項目　**dostounix**

unlink

機能 ファイルを削除します。

形式 int unlink(const char * *filename*);

プロトタイプ dos.h, io.h, stdio.h

解説 **unlink** は，*filename* で指定されたファイルを削除します。*filename* には，DOS のドライブ名，パス名，ファイル名を使うことができます。ワイルドカードは使えません。
読み出し専用ファイルは，**unlink** の呼び出しでは削除できません。読み出し専用ファイルを削除するには，先に **chmod** あるいは **_chmod** を使って読み出し専用属性を変更しておく必要があります。

戻り値 **unlink** は，成功した場合は0を返します。エラーの場合は-1を返し，***errno*** に次のいずれかをセットします。

ENOENT	パス名あるいはファイル名が見つからない
EACCES	アクセスが拒否された

可搬性 **unlink** は UNIX システムで使用できます。

関連項目 **chmod**, **remove**

unlock

機能　ファイルシェアリングロックを解きます。

形式　int unlock(int *handle*, long *offset*, long *length*)；

プロトタイプ　io.h

解説　**unlock** は，DOS 3.*x* のファイルシェアリング機能へのインターフェースを提供します。
unlock は，以前の **lock** の呼び出しでセットされたロックを解除します。エラーを避けるためには，ファイルをクローズする前にロックを解除しておかなければなりません。もちろん，プログラムが終了する前にロックを解除しておかなくてはなりません。

戻り値　**unlock** は，成功した場合は0を返し，エラーの場合は-1を返します。

可搬性　**unlock** は MS-DOS 3.*x* に特有の関数です。これより前のバージョンでは，この関数はサポートされません。

関連項目　**lock, sopen**

va_...

機能　可変個の引数リストを実現します。

形式

```
# include <stdarg.h>
void va_start(va_list param, lastfix) ;
type va_arg(va_list param, type) ;
void va_end(va_list param) ;
```

プロトタイプ　stdarg,h

解説　**vfprintf**, **vprintf** などのいくつかのC関数は，固定の引数リストに加えて，可変個の引数リストをとることができます。**va_...** マクロは，こうした引数リストをアクセスする可搬性のある方法を提供します。これらのマクロは，呼び出された関数側では受け取った引数の個数と型がわからない場合に，引数リストを1つずつ進めて行くために使われます。

ヘッダファイル stdarg.h では，1つの型(**va_list**)と3つのマクロ(**va_start**, **va_arg**, **va_end**) が宣言されています。

va_list

この配列は，**va_arg** と **va_end** が必要とする情報を含んでいます。呼び出された関数が可変の引数リストをとるときは，その関数は **va_list** 型の変数 *param* を宣言します。

va_start

このルーチン（マクロとしてインプリメントされています）は，その関数に渡される可変数の引数の先頭を *param* が指すようにセットします。**va_start** は，**va_arg** や **va_end** が使用される前に使われなければなりません。**va_start** は2つの引数 *param* と *lastfix* をとります(*param* については上の **va_list** を参照してください)。*lastfix* は固定の引数リストの最後の引数の名前です。

va_arg

このルーチン（これもマクロ）は，次に渡されるべき引数（可変引数の1つ）と同じ型，同じ値の式に展開されます。**va_arg** が最初に使用されると，リストの最初の引数が返されます。**va_arg** が使用されるごとに，リストの次の引数が返されます。これは，まず *param* を再参照し，その後 *param* をインクリメントして次の項を指すようにします。**va_arg** は，*type* を，デリファレンスと次の項の位置づけに使います。以後，**va_arg** が呼び出されるたびに，リスト中の次の引数を *param* が指すように変更します。

va_end

このマクロは，呼び出された関数が通常のリターンを行なえるようにします。**va_end** は *param* を変更して，**va_start** が再び呼び出されなければ使えないようにします。**va_arg** がすべての引数を読み終わった後で，**va_end** が呼ばれるようにすべきです。そうでない場合は，プログラム中で何か予測できないことを引き起こす可能性があります。

戻り値　**va_start** と **va_end** は値を返しません。**va_arg** はリスト中の現在の引数（*param* が指しているもの）を返します。

可搬性　**va_arg**, **va_start**, **va_end** は UNIX システムで使用できます。

関連項目　**v...scanf**, **v...printf**

例1

```
#include <stdio.h>
#include <stdarg.h>

/* 0で終わるリストの合計を求める */

void sum(char *msg, ...)
{
   int total = 0;
   va_list ap;
   int arg;

   va_start(ap, msg);
   while ((arg = va_arg(ap,int)) != 0)
      total += arg;
   printf(msg, total);
}

main()
{
   sum("The total of 1+2+3+4 is %d¥n", 1, 2, 3, 4, 0);
}
```

プログラム出力

```
The total of 1+2+3+4 is 10
```

例2

```
#include <stdio.h>
#include <stdarg.h>

void error(char *format, ...)
{
   va_list argptr;

   printf("error: ");
   va_stsrt(argptr, format);
   vprintf(format, argptr);
   va_end (argptr);
}

main()
{
   int value = -1;

   error("this is just an error message¥n");
   error("invalid value %d encountered¥n", value);
}
```

プログラム出力

```
error:this is just an error message
error:in valid value -1 encoutered
```

vfprintf

機能　ストリームに書式つき出力を書き込みます。

形式　# include <stdio.h>

int vfprintf(FILE * *stream*, const char * *format*, va_list *arglist*)；

プロトタイプ　stdio.h

解説　**v...printf** 関数は，**...printf** 関数の別のエントリポイントとして知られています。これらは，対応する **...printf** とほとんど同じように動作しますが，**v...printf** 関数は，引数リストではなく，引数リストへのポインタを受け入れます。

vfprintf は，引数の並びへのポインタ *arglist* を受け取り，*format* によって指される書式文字列中の書式指定を各引数に適用して，書式化されたデータを指定されたストリームに出力します。書式指定の数は，引数と同じだけなければなりません。

書式指定に関する詳細な情報については，**printf** の解説を参照してください。

戻り値　**vfprintf** は，出力したバイト数を返します。エラーの場合は EOF を返します。

可搬性　**vfprintf** は UNIX システム V で使用でき，ANSI C と互換性があります。

関連項目　**printf, va_...**

例　**printf** を参照してください。

vfscanf

機能　ストリームから書式つき入力を行ないます。

形式　# include <stdio.h>

int vfscanf(FILE * *stream*, const char * *format*, va_list *arglist*) ;

プロトタイプ　stdio.h

解説　**v...scanf** の関数は, **...scanf** 関数への別のエントリポイントとして知られています。これらは, 対応する **...scanf** とほとんど同じように動作しますが, **v...scanf** 関数は, 引数リストではなく, 引数リストへのポインタを受け入れます。

vfscanf は, 一連の入力フィールドをスキャンして, 一度に1文字ずつストリームから文字を読み込みます。次に, 引数 *format* によって指される書式文字列中の書式指定にしたがって, 各フィールドを書式化します。最後に, *arglist* が指しているリスト中の各アドレスに, 書式化した入力を格納していきます。書式文字列中の書式指定の個数は, *arglist* が指すリスト中のアドレスの数と同じでなければなりません。

書式指定も含めて, 詳細な情報については **scanf** の解説を参照してください。

vfscanf は, 通常のフィールドの終了（ホワイトスペース）に達する前に, フィールドのスキャンをやめる場合があります。また, いくつかの理由から入力を終了してしまうこともあります。こうした問題については **scanf** の解説を参照してください。

戻り値　**vfscanf** は, 正しくスキャンし, 変換し, 格納した入力フィールドの数を返します。戻り値には, 値を格納しなかった入力フィールドの数は含まれません。値を格納したフィールドがなかった場合は, 戻り値は0になります。

可搬性　**vfscanf** は UNIX システム V で使用できます。

関連項目　**fscanf, scanf, va_...**

vprintf

機能 *stdout* に書式つき出力を書き出します。

形式

include <stdarg.h>
int vprintf(const char * *format*, *va_list arglist*) ;

プロトタイプ stdio.h

解説 **v...printf** 関数は，**...printf** 関数の別のエントリポイントとして知られています。これらは，対応する **...printf** とほとんど同じように動作しますが，**v...printf** 関数は，引数リストではなく，引数リストへのポインタを受け入れます。

vprintf は，引数の並びへのポインタ *arglist* を受け取り，*format* によって指される書式文字列中の書式指定を各引数に適用して，書式化されたデータを *stdout* に出力します。書式指定の数は，引数と同じだけなければなりません。

書式指定に関する詳細な情報については，**printf** の解説を参照してください。

戻り値 **vprintf** は，出力したバイト数を返します。エラーの場合は EOF を返します。

可搬性 **vprintf** は UNIX システム V で使用でき，ANSI C と互換性があります。

関連項目 **printf**, **va_...**

例 **printf** を参照してください。

vscanf

機能 *stdin*から書式つき入力を行ないます。

形式 int vscanf(const char * *format*, *va_list arglist*);

プロトタイプ stdio.h

解説 **v...scanf**の関数は，**...scanf**関数への別のエントリポイントとして知られています。これらは，対応する**...scanf**とほとんど同じように動作しますが，**v...scanf**関数は，引数リストではなく，引数リストへのポインタを受け入れます。

vscanfは，一連の入力フィールドをスキャンして，一度に1文字ずつ*stdin*から文字を読み込みます。次に，引数*format*によって指される書式文字列中の書式指定にしたがって，各フィールドを書式化します。最後に，*arglist*が指しているリスト中の各アドレスに，書式化した入力を格納していきます。書式文字列中の書式指定の個数は，*arglist*が指すリスト中のアドレスの数と同じでなければなりません。

書式指定も含めて，詳細な情報については**scanf**の解説を参照してください。

戻り値 **vscanf**は，正しくスキャンし，変換し，格納した入力フィールドの数を返します。戻り値には，値を格納しなかった入力フィールドの数は含まれません。値を格納したフィールドがなかった場合は，戻り値は0になります。

可搬性 **vscanf**はUNIXシステムVで使用できます。

関連項目 **fscanf, scanf, va_...**

vsprintf

機能　文字列へ書式つき出力を書き込みます。

形式　int vsprintf(char * *buffer*, const char * *format*, va_list *arglist*)；

プロトタイプ　stdio.h

解説　**v...printf** 関数は，**...printf** 関数の別のエントリポイントとして知られています。これらは，対応する **...printf** とほとんど同じように動作しますが，**v...printf** 関数は，引数リストではなく，引数リストへのポインタを受け入れます。

vsprintf は，引数の並びへのポインタ *arglist* を受け取り，*format* によって指される書式文字列中の書式指定を各引数に適用して，書式化されたデータを文字列 * *buffer* に書き込みます。書式指定の数は，引数と同じだけなければなりません。

書式指定に関する詳細な情報については，**printf** の解説を参照してください。

戻り値　**vsprintf** は，出力したバイト数を返します。エラーの場合は EOF を返します。

可搬性　**vsprintf** は UNIX システム V で使用でき，ANSI C と互換性があります。

関連項目　**printf**, **va_...**

例　**printf** を参照してください。

vsscanf

機能　文字列から書式つき入力を行ないます。

形式　int vsscanf(const char * *buffer*, const char * *format*, va_list *arglist*)；

プロトタイプ　stdio.h

解説　**v...scanf** の関数は，**...scanf** 関数への別のエントリポイントとして知られています。これらは，対応する **...scanf** とほとんど同じように動作しますが，**v...scanf** 関数は，引数リストではなく，引数リストへのポインタを受け入れます。

vsscanf は，一連の入力フィールドをスキャンして，一度に1文字ずつ文字列 * *buffer* から文字を読み込みます。次に，引数 *format* によって指される書式文字列中の書式指定にしたがって，各フィールドを書式化します。最後に，*arglist* が指しているリスト中の各アドレスに，書式化した入力を格納していきます。書式文字列中の書式指定の個数は，*arglist* が指すリスト中のアドレスの数と同じでなければなりません。

書式指定も含めて，詳細な情報については **scanf** の解説を参照してください。

戻り値　**vsscanf** は，正しくスキャンし，変換し，格納した入力フィールドの数を返します。戻り値には，値を格納しなかった入力フィールドの数は含まれません。値を格納したフィールドがなかった場合は，戻り値は0になります。文字列の終わりを読み込んだ場合には EOF を返します。

可搬性　**vsscanf** は UNIX システム V で使用できます。

関連項目　**fscanf**, **scanf**, **va_...**

_write

機能　ファイルへの書き込みを行ないます。

形式　int _write(int *handle*, void * *buf*, unsigned *len*)；

プロトタイプ　io.h

解説　**_write** は，*buf* が指しているバッファから *len* バイトを，*handle* に結びつけられているファイルあるいはデバイスに書き込みます。*handle* は，**creat**，**open**，**dup**，**dup2** の呼び出しで得られたファイルハンドルです。

_write によって書き込めるのは最大65534バイトです。65535 (0xFFFF) は-1と同じであり，エラーを返すために使用されます。

_write が扱うファイルはすべてバイナリファイルなので，改行文字 (LF) を CR/LF に置き換えることはしません。

実際に書き込まれたバイト数が指定より小さい場合は，エラー（おそらくディスクがいっぱいになった）を意味していると考えるべきです。

ディスク上のファイルに対しては，書き込みは常にファイルポインタの現在位置 (**lseek** を参照) から行なわれます。デバイスに対しては，そのデバイスへバイトが直接送られます。

O_APPEND オプションでオープンされたファイルに対して，**_write** は，書き込みの前にファイルポインタを EOF まで移動させません。

戻り値　**_write** は，書き込んだバイト数を返します。エラーの場合は-1を返し，グローバル変数 ***errno*** に次のいずれかをセットします。

EACCES	アクセスが否定された
EBADF	ファイル番号が正しくない

可搬性　**_write** は MS-DOS 特有のものです。

関連項目　**lseek**, **read**, **write**

write

機能　ファイルに書き込みを行ないます。

形式　int write(int *handle*, void * *buf*, unsigned *len*)；

プロトタイプ　io.h

解説　**write** は, *buf* が指すバッファから *len* バイトを, *handle* に結びつけられているファイルあるいはデバイスに書き込みます。*handle* は, **creat**, **open**, **dup**, **dup2** の呼び出しで得られたファイルハンドルです。
テキストファイルに書き込みを行なうときには, ファイルに実際に書き込まれるバイト数が指定した数を越えることがあります。
write によって書き込めるのは最大65534バイトです。65535 (0xFFFF) は -1と同じであり, これはエラーを返すために使用されます。
テキストファイルでは, 改行文字 (LF) があると, **write** は CR/LF に置き換えて出力します。
実際に書き込まれたバイト数が指定より小さい場合は, エラー（おそらくディスクがいっぱいになった）を意味していると考えるべきです。
ディスク上のファイルに対しては, 書き込みは常にファイルポインタの現在位置 (**lseek** を参照) から行なわれます。デバイスに対しては, そのデバイスへバイトが直接送られます。
O_APPEND オプションでオープンされたファイルに対しては, **write** は, ファイルポインタを EOF まで移動してからデータを書き込みます。

戻り値　**write** は, 書き込んだバイト数を返します。テキストファイルに対する **write** では, 生成された CR の数はカウントされません。エラーの場合は -1を返し, *errno* に次のいずれかをセットします。

EACCES	アクセスが否定された
EBADF	ファイル番号が正しくない

可搬性　**write** は UNIX システムで使用できます。

関連項目　　creat, lseek, open, read, _write

テキストビデオ関数リファレンス

この節にまとめられているのは，テキストウィンドウの制御などを行なう関数です。これらは，その対象の機種（PC-9801シリーズ，あるいはIBM PCとその互換機）でしか動作しませんので注意してください。

この節で解説しているルーチンのほかに，テキストウィンドウに対して入出力を行なう4つの関数，**cprintf**，**cputs**，**getche**，**putch**があります。この4つについては，前の「**Turbo C 標準関数リファレンス**」を見てください。

clreol

機能　テキストウィンドウ内で，行末までを消去します。

形式　void clreol(void)；

プロトタイプ　conio.h

解説　**clreol**は，カレントテキストウィンドウにおいて，カーソルの位置から行末までのすべての文字を消去します。カーソルは移動しません。

戻り値　ありません。

可搬性　**clreol**は，PC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します。

関連項目　**clrscr**，**delline**，**window**

clrscr

機能　テキストウィンドウをクリアします。

形式　void clrscr(void)；

プロトタイプ　conio.h

解説　**clrscr** は，カレントテキストウィンドウをクリアし，カーソルをウィンドウの左上端（1,1）に位置づけます。

戻り値　ありません。

可搬性　**clrscr** は，PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。

関連項目　**clreol**，**delline**，**window**

delline

機能　テキストウィンドウ内で1行を削除します。

形式　void delline(void)；

プロトタイプ　conio.h

解説　**delline** は，カーソルがある行を削除し，その下にあるすべての行を1行上にずらします。**delline** は，現在アクティブなテキストウィンドウに対して実行されます。

戻り値　ありません。

可搬性　**deline** は，PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。

関連項目　**clreol,clrscr,insline,window**

gettext

機能　テキストモード画面中のテキストをメモリに格納します。

形式　int gettext(int *left*, int *top*, int *right*, int *bottom*, void ＊ *destin*)；

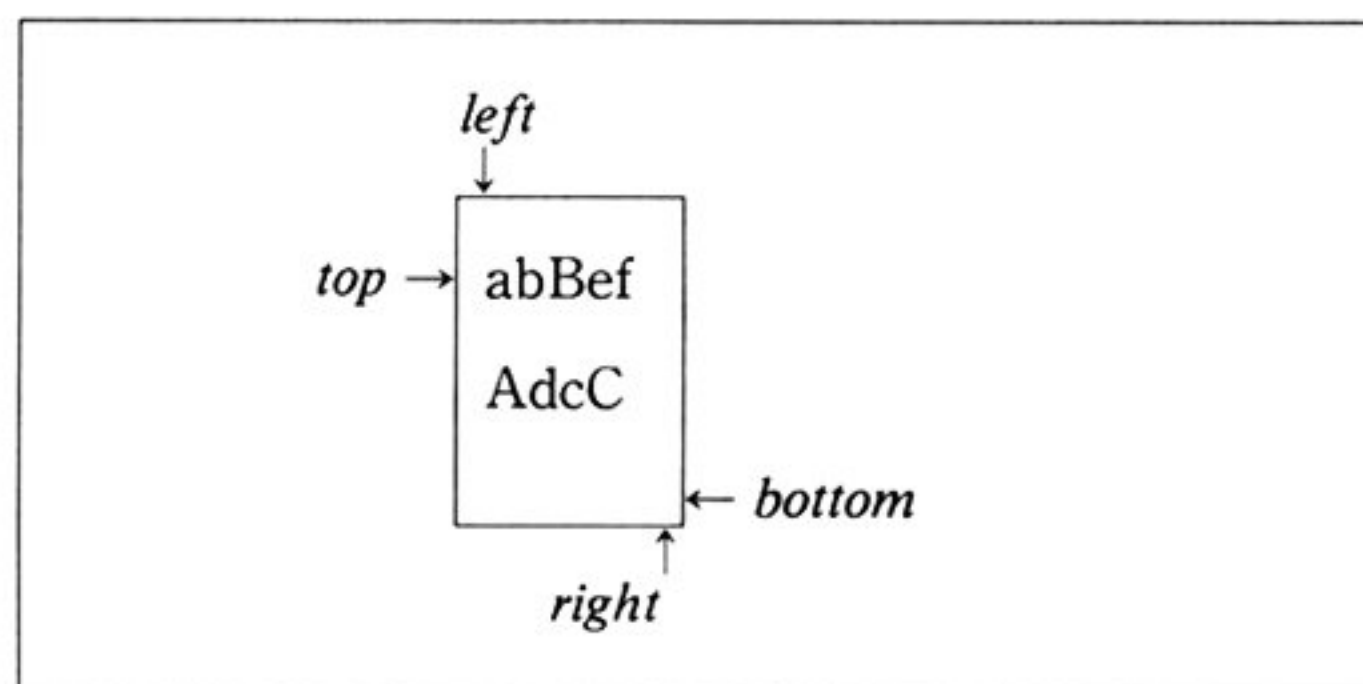
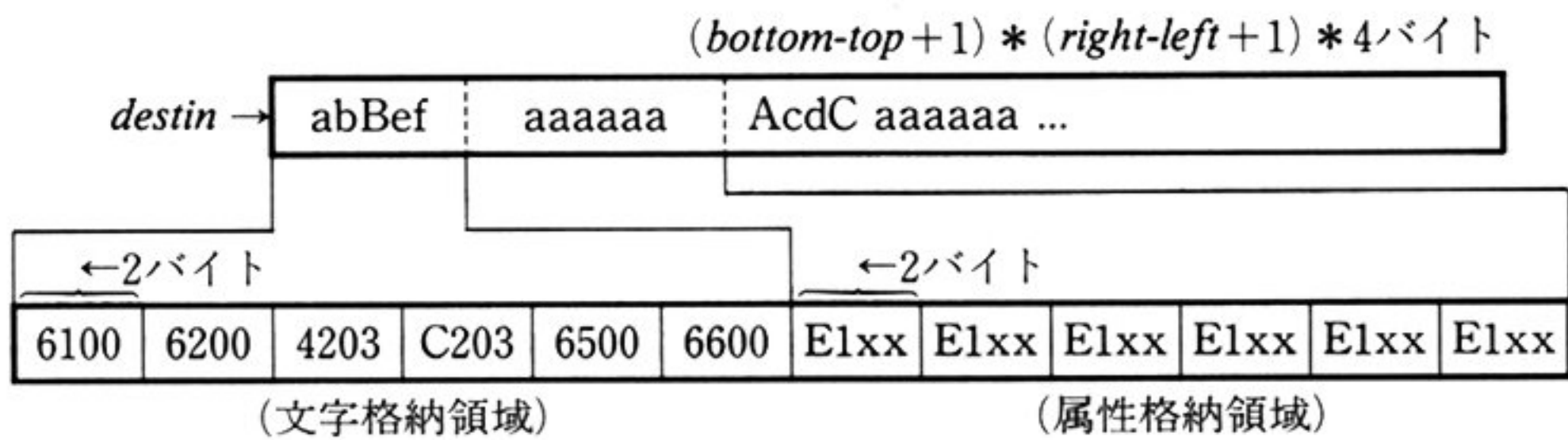
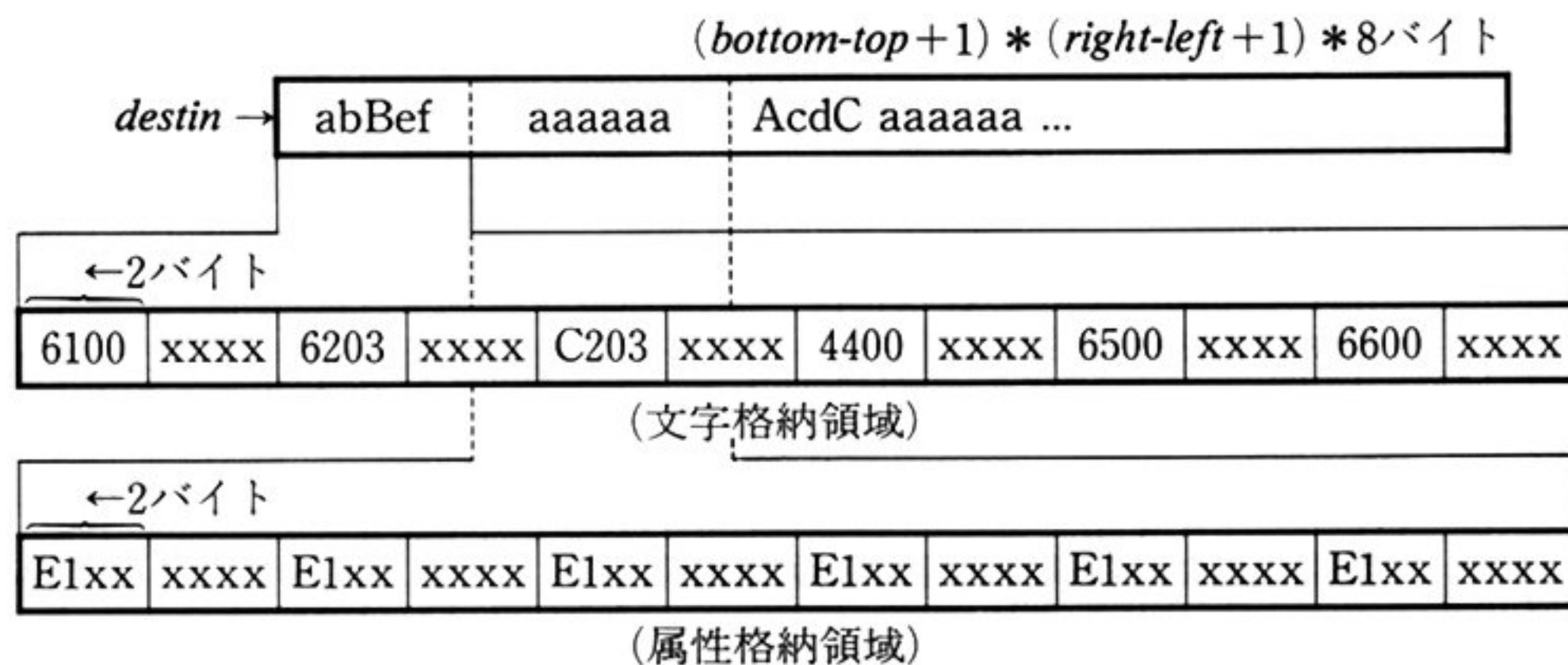
プロトタイプ　conio.h

解説　**gettext** は，始点（*left*,*top*）と終点（*right*,*bottom*）で指定される画面上の長方形の内容を，引数 *destin* が指すメモリ上のバッファに格納します。
4つの座標は，すべて画面絶対座標で指定します(ウィンドウ相対座標ではありません)。画面の左上隅のセルが(1,1)です。
gettext は，画面イメージデータを，左から右へ，また上から下へ順にバッファに格納していきます。

PC-9801では，格納領域 *destin* のサイズ(バイト数)は次のようになり，最低限これだけは確保することが必要です。

モード	行数	桁数	属性
80桁	(*bottom* − *top* ＋ 1)×	(*right* − *left* ＋ 1)×	4
40桁	(*bottom* − *top* ＋ 1)×	(*right* − *left* ＋ 1)×	8

画面イメージと格納領域の関係は次のようになります。

画面イメージ

■ 80 桁モードの場合：

■ 40 桁モードの場合：

E1：文字属性　xx：不定

IBM PC では，画面上の1セルは，メモリ上では2バイトを占めます。第1バイトがセル中の文字，第2バイトがそのビデオ属性です。したがって，画面イメージに対して必要とされる格納領域のサイズは，桁数を w，行数を h とすると，次のようにして求めることができます。

バイト数 ＝ $h \times w \times 2$

戻り値

gettext は，成功したときは1を返します。エラーの場合（たとえば，カレントテキストモードで設定されている行数，桁数を越える範囲が指定された場合など）には0を返します。

可搬性 **gettext** は，PC-9801シリーズ(またはIBM PCとそのBIOSレベルの互換機）でのみ動作します。

関連項目 **movetext**，**puttext**

例

```
#include <conio.h>

main()
{
   char buf [20*10*2];

   gettext(1, 1, 20, 10, buf);     /* 画面の一部をセーブ */

   /* ... */

   puttext(1, 1, buf);             /* 画面を回復 */
}
```

gettextinfo

機能 テキストモードビデオ情報を得ます。

形式 # include <conio.h>

void gettextinfo(struct text_info ＊ *r*)；

プロトタイプ conio.h

解説 **gettextinfo** は，カレントビデオテキスト情報を引数 *r* で指される構造体 **text_info** に格納します。

text_info 構造体は，ヘッダファイル conio.h の中で次のように定義されています。

```
struct text_info {
    unsigned char winleft;       /* ウィンドウの左座標 */
    unsigned char wintop;        /* ウィンドウの上座標 */
    unsigned char winright;      /* ウィンドウの右座標 */
    unsigned char winbottom;     /* ウィンドウの下座標 */
    unsigned char attribute;     /* 現在の文字属性 */
    unsigned char normattr;      /* 通常の文字属性 */
    unsigned char currmode;      /* PC-9801では，
                                      VL/BG 8025,8020,4025,4020
                                    IBM PCでは，
                                      BW80, BW80, C40, C80 */
    unsigned char screenheight;  /* 画面の上限の高さ */
    unsigned char screenwidth;   /* 画面の上限の幅 */
    unsigned char curx;          /* カレントウィンドウ中でのX座標 */
    unsigned char cury;          /* カレントウィンドウ中でのY座標 */
};
```

構造体のメンバ *bank* は現在表示している画面のページ番号で，ウィンドウの大きさ，文字属性，カーソル情報はページ分あります。

戻り値 ありません。ただし結果は *r* が指している構造体に格納されます。

可搬性 **gettextinfo** は，PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。

wherey, window

例

```
#include <conio.h>

void main()
{
    int bank;
    struct text_info tinf;

    gettextinfo(&tinf);         /* 現在の設定を得る */
    if (tinf.currmode & 0x04)
        cprintf("簡易グラフパターン有効モード");
    else
        cprintf("バーチカルライン有効モード");
    if (tinf.currmode & 0x02)
        cprintf("４０桁 × ");
    else
        cprintf("８０桁 × ");
    if (tinf.currmode & 0x01)
        cprintf("２０行¥r¥n");
    else
        cprintf("２５行¥r¥n");
    cprintf("画面の高さ=%d ", tinf.screenheight);
    cprintf("画面の幅=%d¥r¥n", tinf.screenwidth);
    bank = tinf.bank;
    cprintf("ウィンドウの始点 (%d,%d) 終点 (%d,%d) ¥r¥n",
            tinf.winleft[bank], tinf.wintop[bank],
            tinf.winright[bank], tinf.winbottom[bank]);
    cprintf("現在のテキスト表示属性=%02X ",
            tinf.attribute[bank]);
    cprintf("省略時のテキスト表示属性=%02X¥r¥n",
            tinf.normattr[bank]);
    cprintf("カーソル座標 (%d,%d) ¥r¥n",
            tinf.curx[bank], tinf.cury[bank]);
    if (tinf.cursor[bank] & 0x02)
        cprintf("カーソル静止中 ");
    else
        cprintf("カーソル点滅中 ");
    if (tinf.cursor[bank] & 0x01)
        cprintf("カーソル表示中¥r¥n");
    else
        cprintf("カーソル消去中¥r¥n");
}
```

gotoxy

機能　　テキストウィンドウ内のカーソルを位置づけます。

形式　　void gotoxy(int *x*, int *y*);

プロトタイプ　　conio.h

解説　　**gotoxy** は，カレントテキストウィンドウの指定された位置にカーソルを動かします。座標が不適当な場合，**gotoxy** の呼び出しは無視されます。たとえば，(35,25)がウィンドウの右下隅の座標であるのに，**gotoxy**(40,30)と呼び出すことは不可能です。

戻り値　　ありません。

可搬性　　**gotoxy** は，PC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します。これに対応する関数は Turbo Pascal 4.0にもあります。

関連項目　　**wherex**，**wherey**，**window**

例

```
gotoxy(10,20);     /* カーソルを (10,20) に置く */
```

highvideo

機能　高輝度文字を選択します。

形式　void highvideo(void)；

プロトタイプ　conio.h

解説　PC-9801シリーズでは，**highvideo** はなんの動作もしないダミー関数です。

IBM PC では **highvideo** は，現在選択されているフォアグラウンド色の高輝度ビットをセットすることによって，高輝度文字を選択します。
この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および **printf** などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。

戻り値　ありません。

可搬性　PC-9801シリーズでは，IBM PC との互換性のためにエントリだけが用意されています。
IBM PC 版では，IBM PC およびその互換機でのみ動作します。
これに対応する関数は Turbo Pascal にもあります。

関連項目　**lowvideo, normvideo, textattr, textcolor**

insline

機能　テキストウィンドウ内で空行を挿入します。

形式　void insline(void)；

プロトタイプ　conio.h

解説　**insline** は，テキストウィンドウ内のカーソルがある位置に空行を挿入します。空行の下にある行はすべて1行下がり，最下行はウィンドウの下に消えます。

IBM PC では，挿入される行には現在のバックグラウンド色が用いられます。

insline は，テキストモードで使用されます。

戻り値　ありません。

可搬性　**insline** は，PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。これに対応する関数は Turbo Pascal 4.0にもあります。

関連項目　**delline**，**window**

lowvideo

機能　低輝度文字を選択します。

形式　void lowvideo(void)；

プロトタイプ　conio.h

解説　PC-9801シリーズでは，**lowvideo**は何の動作もしないダミー関数です。

IBM PCでは**lowvideo**は，現在選択されているフォアグラウンド色の高輝度ビットをクリアすることによって，低輝度文字を選択します。
この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf**など）が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および**printf**などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。

戻り値　ありません。

可搬性　PC-9801シリーズでは，IBM PCとの互換性のためにエントリだけが用意されています。
IBM PC版では，IBM PCおよびその互換機でのみ動作します。
これに対応する関数はTurbo Pascalにもあります。

関連項目　**highvideo**, **normvideo**, **textattr**, **textcolor**

movetext

機能　画面上のテキスト領域を別の領域にコピーします。

形式　int movetext(int *left*, int *top*, int *right*, int *bottom*,
　　　　int *destleft*, int *desttop)；

プロトタイプ　conio.h

解説　**movetext** は，*left*, *top*, *right*, *bottom* で定義される画面上の長方形領域の内容を，同じ形の別の領域にコピーします。コピー先のブロックの左上隅の位置は（*destleft*,*desttop*）で与えられます。
すべての座標は画面上の絶対座標で表わします。2つの領域がオーバーラップしている場合にも，正しくコピーが行なわれます。
movetext は，直接ビデオ出力を行なうテキストモード関数です。

戻り値　**movetext** は，コピーが成功したときに0以外の値を返し，コピーに失敗したとき（現在の画面モードの範囲を越える座標を指定したときなど）には0を返します。

可搬性　**movetext** は，PC-9801シリーズ(または IBM PC とその BIOS レベルの互換機）でのみ動作します。

関連項目　**gettext**, **puttext**

例

```
/* 左上隅(5,15), 右下隅(20,25)の領域を,
   新しい左上隅(10,20)にコピーする         */
movetext(5, 15, 20, 25, 10, 20);
```

normvideo

機能　画面の表示属性をデフォルトに戻します。

形式　void normvideo(void)；

プロトタイプ　conio.h

解説　**normvideo** は，画面の表示属性の設定を（プログラムが実行を開始したときの）デフォルトの状態に戻します。デフォルトの状態は，構造体 **text_info** の *normattr* にセットされているものです（**gettextinfo** を参照）。
この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および **printf** などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。

戻り値　ありません。

可搬性　**normvideo** は，PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。これに対応する関数は Turbo Pascal にもあります。

関連項目　**highvideo**, **lowvideo**, **textattr**, **textcolor**

puttext

機能　メモリから画面上にテキストをコピーします。

形式　int puttext(int *left*, int *top*, int *right*, int *bottom*, void * *source*)；

プロトタイプ　conio.h

解説　**puttext** は，**gettext** によって格納された *source* が指すメモリ領域の内容を，始点（*left*,*top*）と終点（*right*,*bottom*）で指定される画面上の長方形に復元します。

座標はすべて，画面絶対座標で指定します（ウィンドウ相対座標ではありません）。画面の左上隅が(1,1)です。

puttext は，メモリ領域の先頭から画面上の長方形領域へ，左から右へ，また上から下へ順にテキストを復元します。

戻り値　**puttext** は，成功したときは0以外の値を返します。エラーがあった場合(たとえば，カレントテキストモードで設定されている行数，桁数を越える範囲が指定されたときなど）には0を返します。

可搬性　**puttext** は，PC-9801シリーズ(または IBM PC とその BIOS レベルの互換機）でのみ動作します。

関連項目　**gettext**, **movetext**, **window**

textattr (PC-9801)

機能　　テキスト属性をセットします。

形式　　void textattr(int *newattr*);

プロトタイプ　conio.h

解説　　**textattr** は，1回の呼び出しで文字の色，縦線の有無，下線の有無，反転の指示，点滅の指示などの属性を設定することができます。
この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および **printf** などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。
引数 *newattr* の各ビットの意味は次のようになっています。

b7	b6	b5	b4	b3	b2	b1	b0
G	R	B	VLBG	UL	RV	BK	SC

G	緑の発色（1のとき有効）
R	赤の発色（1のとき有効）
B	青の発色（1のとき有効）
VLBG	**textmode** で VL*xxxx* モードが指定されている場合には縦罫線添付の指示を意味し，BG*xxxx* モードが指定されている場合にはグラフィック文字の表示の指示を意味します（1のとき有効）。
UL	下線添付の指示（1のとき有効）
RV	反転表示の指示（1のとき有効）
BK	点滅表示の指示（1のとき有効）
SC	隠し表示の指示（0のとき有効）

ヘッダファイルconio.h内には下記のように属性が定義されています。

定数名	値	意味
T_BLACK	0x00	黒
T_BLUE	0x20	青
T_GREEN	0x80	緑
T_CYAN	0xA0	水色
T_RED	0x40	赤
T_MAGENTA	0x60	紫
T_BROWN	0xC0	黄色（T_YELLOWと同じ）
T_LIGHTGRAY	0xE0	白（T_WHITEと同じ）
T_DARKGRAY	0x00	黒（T_BLACKと同じ）
T_LIGHTBLUE	0x20	青（T_BLUEと同じ）
T_LIGHTGREEN	0x80	緑（T_GREENと同じ）
T_LIGHTCYAN	0xA0	水色（T_CYANと同じ）
T_LIGHTRED	0x40	赤（T_REDと同じ）
T_LIGHTMAGENTA	0x60	紫（T_MAGENTAと同じ）
T_YELLOW	0xC0	黄色
T_WHITE	0xE0	白
VERTICALLINE	0x10	縦線の添付
NOVERTICALLINE	0x00	
UNDERLINE	0x08	下線の添付
NOUNDERLINE	0x00	
REVERSE	0x04	反転表示
NOREVERSE	0x00	
BLINK	0x02	点滅表示
NOBLINK	0x00	
SECRET	0x00	
NOSECRET	0x01	隠し属性の否定
BASICGRAPH	0x10	グラフィック文字の指示
NOBASICGRAPH	0x00	

戻り値　　ありません。

可搬性　　**textattr** は PC-9801シリーズでのみ動作します。

関連項目　　**gettextinfo, normvideo, textcolor**

例

```
#include  <conio.h>
main()
{
   textmode(VL8025);           /* 対象のモード */
   textattr(CYAN | REVERSE | NOSECRET);
                               /* 水色・反転属性 */
   /* ... */
   textattr(CRED | REVERSE | BLINK | NOSECRET);
                               /* 赤色・反転・点滅属性 */
   /* ... */
}
```

textattr (IBM PC)

機能　テキスト属性をセットします。

形式　void textattr(int *newattr*);

プロトタイプ　conio.h

解説　**textattr**は，1回の呼び出しでフォアグラウンドとバックグラウンドの両方の色をセットすることができます（**textattr**を使わない場合には，これらの属性は，それぞれ**textcolor**および**textbackground**によってセットします)。

この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf**など)が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および**printf**などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。

引数*newattr*の各ビットの意味（色情報）は次のようになっています。

b7	b6	b5	b4	b3	b2	b1	b0
B	b	b	b	f	f	f	f

ffff　この4ビットでフォアグラウンド色を表わします（0～15)。
bbb　この3ビットでバックグラウンド色を表わします（0～8)。
B　ブリンク（点滅）指定ビットです。

ブリンク指定ビットがオンのときには，文字が点滅します。これは，属性値に定数BLINKを加えることによって指定することができます。

*newattr*に指定する値には，conio.hの中で定義されているシンボリック定数を使用することができます。バックグラウンド色の指定に関しては次のような制限があることに注意してください。

■バックグラウンド色には，前半の8色しか選択できません。

■バックグラウンド色は，定数値を左に4ビットシフトすることによって

正しいビット位置が得られます(**例**を参照してください)。

定数名	**値**	**フォア/バック**
BLACK	0	両方
BLUE	1	両方
GREEN	2	両方
CYAN	3	両方
RED	4	両方
MAGENTA	5	両方
BROWN	6	両方
LIGHTGRAY	7	両方
DARKGRAY	8	フォアグラウンドのみ
LIGHTBLUE	9	フォアグラウンドのみ
LIGHTGREEN	10	フォアグラウンドのみ
LIGHTCYAN	11	フォアグラウンドのみ
LIGHTRED	12	フォアグラウンドのみ
LIGHTMAGENTA	13	フォアグラウンドのみ
YELLOW	14	フォアグラウンドのみ
WHITE	15	フォアグラウンドのみ
BLINK	128	フォアグラウンドのみ

戻り値 ありません。

可搬性 **textattr** は IBM PC とその互換機でのみ動作します。

関連項目 **gettextinfo**, **highvideo**, **lowvideo**, **normvideo**, **textbackground**, **textcolor**

例

```
/* バックグラウンドに青, 文字に点滅の黄色を選択する */
textattr(YELLOW + (BLUE<<4) + BLINK);
cputs("Hello, world");
```

textbackground

機能　文字のバックグラウンド色をセットします。

形式　void textbackground(int *newcolor*)；

プロトタイプ　conio.h

解説　PC-9801シリーズでは，**textbackground** は何の動作もしないダミー関数です。

IBM PC では，**textbackground** はバックグラウンドテキスト色を選択します。
この関数の呼び出し以降に，直接ビデオ出力を行なうテキストモード関数によって書かれるすべての文字のバックグラウンドは，*newcolor* で与えられた色になります。*newcolor* に指定するのは0～7の整数です。
textbackground を呼び出した時点で表示されている文字には影響を与えません。
newcolor に指定できる色（シンボリック定数または数値）は次の通りです。

定数名	値
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7

戻り値　ありません。

可搬性　**textbackground**は，PC-9801シリーズではIBM PCとの互換性のためにエントリのみが用意されています。
IBM PCでは，**textbackground**はIBM PCあるいはその互換機でのみ動作します。これに対応する関数はTurbo Pascal 4.0にもあります。

関連項目　**gettextinfo, textattr, textcolor**

例

```
/* バックグラウンドを紫にする */
textbackground(MAGENTA);
```

textbank（PC-9801のみ）

機能　　画面ページを切り換えます。

形式　　void textbank(int *bank*)；

プロトタイプ　conio.h

解説　　**textbank** は，画面ページの切り換えを行ないます。引数 *bank* には0または1を指定してください。

2つの画面ページにはそれぞれ別々に，ウィンドウ，文字属性，カーソル位置の属性をセットすることができます。ただし，テキストモードは同じものになります。

注意：PC-9801の最初期型，およびPC-9801U2には画面ページは1枚しかない(テキストVRAMが1画面分しかない)ので，ページ1を使用することはできません。

戻り値　　ありません。

可搬性　　**textbank** は，PC-9801でのみ動作します。

関連項目　**textmode**

例

```
#include <conio.h>
void main()
{
   textbank(1);                  /* バンク切り換え */
   textcolor(BLUE);
   cprintf("ページ１を表示しました。¥r¥n");
   textbank(0);                  /* バンク切り換え */
   textcolor(GREEN);
   cprintf("ページ０を表示しました。¥r¥n");
}
```

textblink (PC-9801のみ)

機能　　文字の点滅属性をセットします。

形式　　void textblink(int *blink*);

プロトタイプ　　conio.h

解説　　**textblink** は，文字の点滅属性をセットします。引数 *blink* には，conio.h されているシンボリック定数 BLINK (0x02) または NOBLINK (0x00) 指定します。

この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）の文字に影響を与えます。

戻り値　　ありません。

可搬性　　**textblink** は，PC-9801シリーズでのみ動作します。

関連項目　　**textattr**

例

```
#include <conio.h>
void main()
{
   textblink(BLINK);              /* 点滅を設定 */
   cprintf("点滅表示¥r¥n");
   textblink(NOBLINK);            /* 点滅を解除 */
}
```

textcolor (PC-9801)

機能　文字の色属性をセットします。

形式

```
#include <conio.h>
void textcolor(int newcolor);
```

プロトタイプ　conio.h

解説　**textcolor** は，文字の色属性をセットします。

この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および **printf** などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。

newcolor には，conio.h に定義されている以下のシンボリック定数(あるいはその値）を指定します。

定数名	値	意味
T_BLACK	0x00	黒
T_BLUE	0x20	青
T_GREEN	0x80	緑
T_CYAN	0xA0	水色
T_RED	0x40	赤
T_MAGENTA	0x60	紫
T_BROWN	0xC0	黄色（T_YELLOW と同じ）
T_LIGHTGRAY	0xE0	白（T_WHITE と同じ）
T_DARKGRAY	0x00	黒（T_BLACK と同じ）
T_LIGHTBLUE	0x20	青（T_BLUE と同じ）
T_LIGHTGREEN	0x80	緑（T_GREEN と同じ）
T_LIGHTCYAN	0xA0	水色（T_CYAN と同じ）
T_LIGHTRED	0x40	赤（T_RED と同じ）

T_LIGHTMAGENTA	0x60	紫（T_MAGENTAと同じ）
T_YELLOW	0xC0	黄色
T_WHITE	0xE0	白

戻り値　　ありません。

可搬性　　**textcolor** はPC-9801シリーズでのみ動作します。これに対応する関数がTurbo Pascalにもあります。

関連項目　　**gettextinfo, normvideo, textattr, textbackground**

textcolor (IBM PC)

機能 テキストモードでの文字色を選択します。

形式

```
#include <conio.h>
void textcolor(int newcolor);
```

プロトタイプ conio.h

解説 **textcolor** は，フォアグラウンドの文字色を選択します。以降にコンソール出力関数によって書かれるすべての文字のフォアグラウンド色は，*newcolor* に与えられた色になります。*newcolor* には，conio.h で定義されているシンボリック定数を使用することができます。シンボリック定数を使う場合には，conio.h をインクルードしなければなりません。

この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）が表示する文字にのみ影響を与えます。**textcolor** が呼び出された時点で表示されている文字には影響を与えません。また，**printf** などの標準入出力ストリームを使用する関数には影響しません。

次に示したのは，フォアグラウンドに使用できる色（シンボリック定数）およびその値です。

定数名	値	定数名	値
BLACK	0	DARKGRAY	8
BLUE	1	LIGHTBLUE	9
GREEN	2	LIGHTGREEN	10
CYAN	3	LIGHTCYAN	11
RED	4	LIGHTRED	12
MAGENTA	5	LIGHTMAGENTA	13
BROWN	6	YELLOW	14
LIGHTGRAY	7	WHITE	15
		BLINK	128

文字をブリンク（点滅）させたいときには，フォアグラウンド色に128を加えます。これには定義済みの定数 BLINK を使うことができます。たとえば次のようにします。

```
textcolor(CYAN + BLINK);
```

注意：モニタの種類によっては，8つの明るい色（8～15）を作るために用いられる高輝度信号が認識されないことがあります。このようなモニタでは，明るい色は暗い色（0～7）とまったく同じ色になります。また，システムによってはこれらの数値を色の違いではなく，単色の階調，特殊なパターン，あるいは特別な属性（下線，ボールド，イタリック，など）で表現するものもあります。こうしたシステムで，色の指定が実際に画面にどのように表示されるかはハードウェアに依存します。

戻り値　ありません。

可搬性　**textcolor** は，IBM PC およびその互換機でのみ動作します。これに対応する関数が Turbo Pascal にもあります。

関連項目　**gettextinfo**, **highvideo**, **lowvideo**, **normvideo**, **textattr**, **textbackground**

textcursor（PC-9801のみ）

機能　カーソルの属性を設定します。

形式　void textcursor(int *cursor*)；

プロトタイプ　conio.h

解説　**textcursor** は，カーソル属性を設定します。カーソルの表示または非表示の指定，カーソルの静止表示または点滅表示の指定を行ないます。
引数 *cursor* には，ヘッダファイル conio.h で定義されているシンボリック定数を指定することができます。

定数名	値	意味
DISP_CURSOR	0x01	カーソル表示
NODISP_CURSOR	0x00	カーソル非表示
BLINK_CURSOR	0x00	カーソル点滅表示
NOBLINK_CURSOR	0x02	カーソル静止表示

戻り値　ありません。

可搬性　PC-9801シリーズでのみ動作します。

例

```
#include <conio.h>
void main()
{
    textcursor(DISP_CURSOR | BLINK_CURSOR);
                                    /* カーソル点滅表示 */
    textcursor(DISP_CURSOR | NOBLINK_CURSOR);
                                    /* カーソル静止表示 */
    textcursor(NODISP_CURSOR)       /* カーソル非表示   */
    /*...*/
}
```

textmode（PC-9801）

機能　テキストモードを設定します。

形式　void textmode(int *newmode)*；

プロトタイプ　conio.h

解説　**textmode** は，引数 *newmode* で指定されたテキストモードを設定します。目的とするテキストモードは，conio.h に定義されている列挙型 *text modes* の中のシンボリック定数によって指定することができます。シンボリック定数を使用する場合には，conio.h をインクルードしなければなりません。
テキストモードはページごとに異なるモードを設定することはできません（**textbank** を参照）。また，この関数で設定を行なわずに表示関数を使用したとき，桁数と行数，縦罫線の有無，および文字の色属性は，プログラムを起動したときに設定されていた状態と同じものになります。
text_modes 型の定数，数値，および設定されるモードは以下の通りです。

定数名	**値**	**テキストモード**
LASTMODE	-1	直前に設定したテキストモード
VL8025	0	80桁×25行，縦罫線が有効
VL8020	1	80桁×20行，縦罫線が有効
VL4025	2	40桁×25行，縦罫線が有効
VL4020	3	40桁×20行，縦罫線が有効
BG8025	4	80桁×25行，グラフィック文字が有効
BG8020	5	80桁×20行，グラフィック文字が有効
BG4025	6	40桁×25行，グラフィック文字が有効
BG4020	7	40桁×20行，グラフィック文字が有効

VL*xxxx* を設定したときに画面に表示される文字イメージは次のようになります。

VL	ANK コード	シフト JIS コード
VL オフ	半角文字	全角文字
VL オン	縦罫線つき半角文字	縦罫線つき全角文字

注意：VL オンとは，**textvertical** によって縦罫線の表示が有効になっていることをいいます。

BG*xxxx* を設定したときに画面に表示される文字イメージは次のようになります。

BG	ANK コード	シフト JIS コード
VL オフ	半角文字	グラフィック文字（半角）
VL オン	簡易グラフィックパターン	簡易グラフィックパターン

注意：BG*xxxx* の場合は，シフト JIS コードから JIS コードへの変換は行ないません。

textmode が呼び出されると，カレントウィンドウは画面全体にリセットされ，カレントテキスト属性はノーマルにリセットされます。ノーマル属性は **normvideo** によって設定される属性に対応します。

戻り値　　ありません。

可搬性　　**textmode** は PC-9801でのみ動作します。対応する関数が Turbo Pascal にもあります。

関連項目　　**gettextinfo**，**window**

textmode（IBM PC）

機能　テキストモードを設定します。

形式　void textmode(int *newmode*)；

プロトタイプ　conio.h

解説　**textmode** は，指定されたテキストモードを選択します。
目的とするテキストモード(引数 *newmode*)は，conio.h に定義されている列挙型 *text_modes* の中のシンボリック定数によって与えます。これらの定数を使用する場合には，conio.h をインクルードしなければなりません。
text_modes 型の定数，数値，および設定されるモードは以下の通りです。

定数名	値	テキストモード
LASTMODE	-1	直前に設定したテキストモード
BW40	0	白黒，40桁
C40	1	カラー，40桁
BW80	2	白黒，80桁
C80	3	カラー，80桁
MONO	7	モノクロ，80桁

textmode が呼び出されると，カレントウィンドウは画面全体にリセットされ，カレントテキスト属性はノーマルにリセットされます。ノーマル属性は **normvideo** によって設定される属性に対応します。
LASTMODE を指定して **textmode** を呼び出すと，前回選択されたテキストモードに再び設定されます。この機能は，グラフィックスモードを使用した後でテキストモードに戻るような場合にのみ有用となります。

textmode は，画面がテキストモードである(つまり現在のテキストモードを別のモードに変更する）場合にのみ使用されるべきです。画面がグラフィックスモードである場合には，**restorecrtmode** によってテキストモードに戻ってから，**textmode** を呼び出すようにしてください。

戻り値 ありません。

可搬性 **textmode** は，IBM PC およびその互換機でのみ動作します。これに対応する関数が Turbo Pascal にもあります。

関連項目 **gettextinfo**，**window**

textreverse（PC-9801のみ）

機能　文字の反転属性をセットします。

形式　# include <conio.h>
void textreverse(int *reverse)；

プロトタイプ　conio.h

解説　**textreverse** は，文字の反転属性をセットします。引数 *reverse* には，conio.h に定義されているシンボリック定数 REVERSE（0x04）または NOREVERSE（0x00）を指定します。
この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および **printf** などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。

戻り値　ありません。

可搬性　PC-9801シリーズでのみ動作します。

関連項目　**textattr**

例

```
#include <conio.h>
void main()
{
    textreverse(REVERSE);          /* 反転を設定 */
    cprintf("反転表示¥r¥n");
    textreverse(NOREVERSE);        /* 反転を解除 */
}
```

textunder（PC-9801のみ）

機能　文字の下線属性をセットします。

形式　# include <conio.h>
void textunder(int *under)*；

プロトタイプ　conio.h

解説　**textunder** は，文字の下線属性をセットします。引数 *under* には，conio.h に定義されているシンボリック定数 UNDERLINE (0x08) または NOUNDERLINE (0x00) を指定します。
この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および **printf** などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。

戻り値　ありません。

可搬性　PC-9801シリーズでのみ動作します。

関連項目　**textattr**

例

```
#include <conio.h>
void main()
{
   textunder(UNDERLINE);              /* 下線を設定 */
   cprintf("下線表示¥r¥n");
   textunder(NOUNDERLINE);            /* 下線を解除 */
}
```

textvertical（PC-9801のみ）

機能　文字の縦線属性をセットします。

形式　# include <conio.h>
void textvertical(int *vertical*);

プロトタイプ　conio.h

解説　**textvertical** は，文字の縦線属性をセットします。引数 *vertical* には，conio.h で定義されている VERTICAL（0x10）または NOVERTICAL（0x00）を指定します。

この関数の呼び出しは，以降に呼び出される直接コンソール入出力関数（**cprintf** など）が出力する文字にのみ影響を与えます。すでに画面に表示されている文字，および **printf** などの標準入出力ストリームを使用する関数が出力する文字には影響を与えません。

textvertical は，テキストモードが BGxxxx の場合には，グラフイック文字の表示切り換えの意味になります。

戻り値　ありません。

可搬性　PC-9801シリーズでのみ動作します。

関連項目　**textattr**，**textmode**

例

```
#include <conio.h>
void main()
{
   textvertical(VERTICALLINE);          /* 縦線を設定 */
   cprintf("縦線表示¥r¥n");
   textvertical(NOVERTICALLINE);        /* 縦線を解除 */
}
```

wherex

機能 ウィンドウ内での水平方向のカーソル位置を得ます。

形式 int wherex(void);

プロトタイプ conio.h

解説 **wherex** は，カレントテキストウィンドウ内でのカーソル位置の x 座標を返します。

戻り値 **wherex** は，1～80の範囲の整数値を返します。

可搬性 **wherex** は，PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。これに対応する関数は Turbo Pascal にもあります。

関連項目 **gettextinfo**, **gotoxy**, **wherey**

例

```
printf(" The cursor is at (%d,%d)¥n", wherex(), wherey());
```

wherey

機能　ウィンドウ内での垂直方向のカーソル位置を得ます。

形式　int wherey(void);

プロトタイプ　conio.h

解説　**wherey** は，カレントテキストウィンドウ内でのカーソル位置の y 座標を返します。

戻り値　**wherey** は，1～25の範囲の整数値を返します。

可搬性　**wherey** は，PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。これに対応する関数は Turbo Pascal にもあります。

関連項目　**gettextinfo**, **gotoxy**, **wherex**

例　**wherex** を参照してください。

window

機能　アクティブテキストウィンドウを定義します。

形式　void window(int *left*, int *top*, int *right*, int *bottom*);

プロトタイプ　conio.h

解説　**window** は，画面上にテキストウィンドウを定義します。座標が有効でない場合は，**window** の呼び出しは無視されます。

left と *top* はウィンドウの左上隅画面上の座標です。

right と *bottom* は右下隅の画面上の座標です。

テキストウィンドウの最小の大きさは，1行1桁です。デフォルトウィンドウは画面全体であり，次のような座標を持っています。

■ PC-9801：

80桁25行モード	(1,1)－(80,25)
80桁20行モード	(1,1)－(80,20)
40桁25行モード	(1,1)－(40,25)
40桁20行モード	(1,1)－(40,20)

■ IBM PC：

80桁モード	(1,1)－(80,25)
40桁モード	(1,1)－(40,25)

戻り値　ありません。

可搬性　**window** は，PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。これに対応する関数は Turbo Pascal にもあります。

関連項目　**clreol**, **clrscr**, **delline**, **gettextinfo**, **gotoxy**, **insline**, **puttext**, **textmode**

グラフィックス関数リファレンス

arc

機能　円弧を描きます。

形式　# include <graphics.h>
void far arc(int *x*, int *y*, int *stangle*, int *endangle*, int *radius*);

プロトタイプ　graphics.h

解説　**arc** は，円弧をカレントドロウカラーで描きます。
arc は，中心が(*x*, *y*)，半径 *radius* の円弧を描きます。円弧は *stangle* から *endangle* までです。*stangle*＝0かつ *endangle*＝360の場合は，**arc** の呼び出しは完全な円を描きます。
arc の角度は，反時計まわりで，0度が3時の方向，90度が12時の方向です。

注意：カレントラインスタイルは，円弧には影響を与えませんが，ライン幅は円弧に影響します。

注意[IBM PC]：CGA のハイレゾリューションモード，あるいはモノクログラフィックスアダプタを使用している場合，このマニュアルにあるサンプルプログラムは期待通りに動かないことがあります。CGA のハイレゾリューションモードまたはモノクロアダプタを使っている場合は，フィルカラーあるいはドロウカラーをセットする関数（**setcolor, setfillstyle, setlinestyle** など）には，シンボリック定数を使わずに，値1を渡すようにしてください。この後の**例 2** で，CGA あるいはモノクロアダプタ上で，**arc, circle, ellipse, getarccoords, getaspectratio,** および **pieslice** をどのように使うかを示しています。

戻り値　ありません。

可搬性　arcはPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**circle, ellipse, fillellipse, getarccordes, pieslice, sector**

例1

```
#include <graphics.h>
#include <conio.h>

main()
{
   int graphdriver = DETECT, graphmode;    /* 自動検出の要求 */
   struct arccoordstype arcinfo;
   int xasp, yasp;
   long xlong;

   initgraph(&graphdriver, &graphmode, "");    /* 初期化 */

   /* 中心角90度で半径50の円弧 */
   arc(150, 150, 0, 89, 50);

   /* 円弧の座標を得て，弦を結ぶ */
   getarccoords(&arcinfo);
   line(arcinfo.xstart, arcinfo.ystart,
      arcinfo.xend, arcinfo.yend);

   /* 円を描く */
   circle(150, 150, 100);

   /* 円の中に楕円を描く */
   ellipse(150, 150, 0, 359, 100, 50);

   /* 扇型を描いてフィルする */
   setcolor(WHITE);                      /* 輪郭は白 */
   setfillstyle(SOLID_FILL, LIGHTRED);
   pieslice(100, 100, 0, 135, 49);
   setfillstyle(SOLID_FILL, LIGHTBLUE);
   pieslice(100, 100, 135, 225, 49);
   setfillstyle(SOLID_FILL, WHITE);
   pieslice(100, 100, 225, 360, 49);

   /* 長方形を描く */
   getaspectratio(&xasp, &yasp);
   xlong = (100L * (long)yasp) / (long)xasp;
   rectangle(0, 0, (int)xlong, 100);
   getch();
   closegraph();
}
```

例 2

```
#include <graphics.h>
#include <conio.h>

main()
{
    int     graphdriver = DETECT, graphmode;    /* 自動検出の要求 */
    struct arccoordstype arcinfo;
    int     xasp, yasp;
    long    xlong;

    initgraph(&graphdriver, &graphmode, "");    /* 初期化 */

    /* 中心角90度で半径50の円弧 */
    arc( 100, 120, 0, 89, 50 );

    /* 円弧の座標を得て，弦を結ぶ */
    getarccoords( &arcinfo );
    line(arcinfo.xstart, arcinfo.ystart,
         arcinfo.xend, arcinfo.yend);

    /* 円を描く */
    circle(100, 120, 80);

    /* 円の中に楕円を描く */
    ellipse(100, 120, 0, 359, 80, 20);

    /* 扇型を描いてフィルする */
    setfillstyle(HATCH_FILL, 1);
    pieslice(200, 50, 0, 134, 49);
    setfillstyle(SLASH_FILL, 1);
    pieslice(200, 50, 135, 225, 49);
    setfillstyle(WIDE_DOT_FILL, 1);
    pieslice(200, 50, 225, 360, 49);

    /* 長方形を描く */
    getaspectratio(&xasp, &yasp);
    xlong = (50L * (long) yasp) / (long) xasp;
    rectangle(0, 0, (int)xlong, 50);
    getch();
    closegraph();
}
```

bar

機能　二次元バー（長方形）を描きます。

形式　# include <graphics.h>
void far bar(int *left*, int *top*, int *right*, int *bottom*)；

プロトタイプ　graphics.h
include <conio.h>

解説　**bar** は，フィルされた二次元バー(長方形)を描きます。バーは，カレントフィルパターンとフィルカラーを使ってフィルされます。**bar** は長方形の輪郭は描きません。輪郭のある2次元のバーを描くには，*depth* を0にして **bar3d** を用いてください。
長方形の左上と右下は，（*left, top*）と（*right*, *bottom*）で与えます。これらの座標は，ピクセルを参照します。

戻り値　ありません。

可搬性　**bar** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**bar3d**, **rectangle**, **setcolor**, **setfillstyle**

例

```
#include <graphics.h>

main()
{
   int graphdriver = DETECT, graphmode;   /* 自動検出を要求 */

   initgraph(&graphdriver, &graphmode, "");    /* 初期化 */
   setfillstyle(SOLID_FILL, MAGENTA);
   bar3d(100, 10, 200, 100, 5, 1);
   setfillstyle(HATCH_FILL, RED);
   bar(30, 30, 80, 80,);

   getche();
   closegraph();
}
```

bar3d

機能　3次元バーを描きます。

形式　# include <graphics.h>
void far bar3d(int *left*, int *top*, int *right*, int *bottom*,
　　int *depth*, int *topflag*)；

プロトタイプ　graphics.h

解説　**bar3d** は，3次元のバーを描き，カレントフィルパターンとフィルカラーで中をフィルします。バーの3次元の輪郭は，カレントラインスタイルとカラーで描きます。バーの奥行きは，ピクセル数で，*depth* によって与えます。*topflag* は，バーに3D トップをつけるかどうかを指定するために使います。*topflag* が0でなければ，トップがつけられます。そうでなければ，(バーを積み重ねることを可能とするために) バーにトップをつけません。**bar3d** は，長方形の左上と右下を，(*left*, *top*) と (*right*, *bottom*) で与えます。**bar3d** の典型的な奥行きは，次のようにバーの幅の25%をとればよいでしょう。

```
bar3d(left, top, right, bottom, (right - reft)/4, 1);
```

戻り値　ありません。

可搬性　**bar3d** は PC-9801シリーズ (または IBM PC とその互換機) でのみ動作します (IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**bar**, **rectangle**, **setcolor**, **setfillstyle**, **setlinestyle**

例　**bar** を参照してください。

circle

機能　与えられた半径で指定の点を中心に円を描きます。

形式　# include <graphics.h>
void far circle(int *x*, int *y*, int *radius*)；

プロトタイプ　graphics.h

解説　**circle** は，カレントドロウカラーで，中心(*x*,*y*)に，半径 *radius* の円を描きます。

注意：カレントラインスタイルは，円周には影響を与えません。ただしライン幅は円周に影響します。

各グラフィックスドライバとグラフィックスモードには，アスペクト比(ピクセルの縦横比)が含まれています。**circle** は，画面に弧を描く際のスケーリングとしてこのアスペクト比を使います。使用しているディスプレイで円が真円として描かれない場合には，**setaspectratio** によって，アスペクト比を調整することができます。

戻り値　ありません。

可搬性　**circle** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**arc**, **ellipse**, **fillellipse**, **getaspectratio**, **pieslice**, **sector**, **setaspectratio**

例　**arc** を参照してください。

cleardevice

機能　グラフィックス画面をクリアします。

形式

```
#include <graphics.h>
void far cleardevice(void);
```

プロトタイプ　graphics.h

解説　**cleardevice** は，グラフィックス画面全体をクリアし(つまりカレントバックグラウンドカラーでフィルします)，CP(カレントポジション)をホームポジション（0,0）に移します。

戻り値　ありません。

可搬性　**cleardevice** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**clearviewport**

clearviewport

機能　カレントビューポートをクリアします。

形式　# include <graphics.h>
void far clearviewport(void)；

プロトタイプ　graphics.h

解説　**clearviewport** は，ビューポートをクリアし，CP（カレントポジション）をビューポート相対のホームポジション（0,0）に移します。

戻り値　ありません。

可搬性　**clearviewport** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**cleardevice**，**getviewsettings**，**setviewport**

例

```
#include <graphics.h>

main()
{
   int graphdriver = DETECT, graphmode;  /* 自動検出を要求 */

   setviewport(30, 30, 130, 130, 0);
   outtextxy(10, 10, "Het any key to clear viewport ...");
   getch();             /* キー入力待ち */
   clearviewport();     /* キー入力があったらビューポートをクリア */
   closegraph();
}
```

closegraph

機能　グラフィックスシステムの使用を終了します。

形式　# include <graphics.h>
void far closegraph(void)；

プロトタイプ　graphics.h

解説　**closegraph** は，グラフィックスシステムによって割り当てられたメモリを解放し，**initgraph** を呼び出す前のモードに画面を戻します(グラフィックスシステムは，**_graphfreemem** を呼び出して，ドライバ，フォント，内部バッファなどに使用していたメモリを解放します)。

戻り値　ありません。

可搬性　**closegraph** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**initgraph**，**setgraphbufsize**

detectgraph

機能　ハードウェアをチェックして，使用すべきグラフィックスドライバとモードを決定します。

形式

```
#include <graphics.h>
void far detectgraph(int far * graphdriver, int far * graphmode) ;
```

プロトタイプ　graphics.h

解説　**detectgraph** は，使用しているシステムに装備されているグラフィックスハードウェアを検出し，最高の品質が得られるグラフィックスドライバとモードを決定します。

PC-9801では，GRCG あるいは EGC が備えられているかどうかを調べ，最も高速な出力，および4096色中16色が使用可能か（あるいは8色のみか）どうかを検出します。

IBM PC では，どのグラフィックスアダプタが装着されているかを調べ，そのアダプタで最も高い解像度が得られるモードを選択します。グラフィックスアダプタがなにも検出できない場合には，* *graphdriver* に-2がセットされ，**graphresult** は-2を返します。

＊ *graphdriver* は，使用すべきグラフィックスドライバを示す整数です。これには，graphics.h で定義されている次のような列挙型 *graphics_drivers* の定数を与えることができます。

graphics_drivers 定数	数値
DETECT	0（自動検出要求）
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10
PC98	11（PC-9801のみ）
PC98GRCG	12（PC-9801のみ）
PC98EGC	13（PC-9801のみ）

* *graphmode* は，初期グラフィックスモードを示す整数です（ただし，* *graphdriver* が DETECT に等しい場合には，* *graphmode* は検出されたドライバにおいて最も高い解像度が得られるモードにセットされます）。* *graphmode* には，graphics.h で定義されている以下のような列挙型 *graphics_modes* の定数を与えることができます。

ドライバ	*graphics_modes*	値	横×縦	パレット	ページ
PC98	PC98C8	0	640×400	8色	2
	PC98C16	1	640×400	16色	2
PC98GRCG	PC98C8	0	640×400	8色	2
	PC98C16	1	640×400	16色	2
PC98EGC	PC98C8	0	640×400	8色	2
	PC98C16	1	640×400	16色	2
CGA	CGAC0	0	320×200	C0	1
	CGAC1	1	320×200	C1	1
	CGAC2	2	320×200	C2	1
	CGAC3	3	320×200	C3	1
	CGAHI	4	640×200	2色	1
MCGA	MCGAC0	0	320×200	C0	1
	MCGAC1	1	320×200	C1	1
	MCGAC2	2	320×200	C2	1
	MCGAC3	3	320×200	C3	1
	MCGAMED	4	640×200	2色	1
	MCGAHI	5	640×480	2色	1
EGA	EGALO	0	640×200	16色	4
	EGAHI	1	640×350	16色	2

EGA64	EGA64LO	0	640×200	16色	1
	EGA64HI	1	640×350	4色	1
EGAMONO	EGAMONOHI	3	640×350	2色	1*
	EGAMONOHI	3	640×350	2色	2**
HERC	HERCMONOHI	0	720×348	2色	2
ATT400	ATT400C0	0	320×200	C0	1
	ATT400C1	1	320×200	C1	1
	ATT400C2	2	320×200	C2	1
	ATT400C3	3	320×200	C3	1
	ATT400MED	4	640×200	2色	1
	ATT400HI	5	640×400	2色	1
VGA	VGALO	0	640×200	16色	2
	VGAMED	1	640×350	16色	2
	VGAHI	2	640×480	16色	1
PC3270	PC3270	0	720×350	2色	1
IBM8514	IBM8514HI	0	640×480	256色	2
	IBM8514LO	1	024×768	256色	2

*　64K EGA モノカード

** 256K EGA モノカード

注意：**detectgraph** を直接呼び出す主な理由は，**detectgraph** が **initgraph** に対して推奨するグラフィックスモードを変更することにあります。

戻り値　ありません。

可搬性　**detectgraph** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します。

関連項目　**graphresult**，**initgraph**

drawpoly

機能　　多角形の輪郭を描きます。

形式

```
#include <graphics.h>
void far drawpoly(int numpoints, int far * polypoints);
```

プロトタイプ　graphics.h

解説　　**drawpoly** は，頂点の数が *numpoints* の多角形を，カレントラインスタイルとカラーで描きます。

polypoints は，(*numpoints*×2個の) 整数の並びを指します。整数の各ペアは，それぞれ多角形の頂点の *x* 座標と *y* 座標を与えます。

注意：*n* 個の頂点からなる閉じた図形を描くためには，*n* 番目の座標が0番目の座標に等しい *n*+1組の座標を，**drawpoly** に渡さなければなりません。

戻り値　　多角形を描いているときにエラーが発生した場合，**graphresult** は-6を返します。

可搬性　　**drawpoly** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　　**fillpoly**，**floodfill**，**graphresult**，**setwritemode**

例

```
#include <graphics.h>
#include <conio.h>

main()
{
   int graphdriver = DETECT, graphmodc;    /* 自動検出を要求 */
   int triangle[ ] = {50,100, 100,100, 150,150, 50,100};
   int rhombus[ ] = {50,10, 90,50, 50,90, 10,50};

   initgraph(&graphdriver, &graphmode, "");  /* 初期化 */

   /* 三角形を描く */
   drawpoly(sizeof(triangle)/(2*sizeof(int)),triangle);

   /* 平行四辺形を描いてフィルする */
   fillpoly(sizeof(rhombus) / (2*sizeof(int)), rhombus);

   getche();
   closegraph();
}
```

ellipse

機能　楕円弧を描きます。

形式

```
#include <graphics.h>
void far ellipse(int x, int y, int stangle, int endangle,
                 int xradius, int yradius);
```

プロトタイプ　graphics.h

解説　**ellipse**は，中心が(*x*,*y*)，水平軸と鉛直軸がそれぞれ*xradius*と*yradius*で与えられる楕円弧をカレントドロウカラーで描きます。弧は*stangle*から*endangle*までです。*stangle*＝0および*endangle*＝360の場合は，**ellipse**の呼び出しは完全な楕円を描きます。

ellipseに与える角度は反時計まわりで，0度が時計で3時の方向，90度が12時の方向になります。

注意：カレントラインスタイルは楕円弧には影響を与えません。ただし，ライン幅は楕円弧に影響します。

戻り値　ありません。

可搬性　**ellipse**はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**arc**，**circle**，**fillellipse**，**getaspectratio**，**pieslice**，**sector**，**setaspectratio**

例　**arc**を参照してください。

fillellipse

機能　楕円を描いてフィルします。

形式　#include <graphics.h>
void far fillellipse(int *x*, int *y*, int *xradius*, int *yradius*)；

プロトタイプ　graphics.h

解説　**fillellipse** は，中心が(*x*,*y*)，水平軸と鉛直軸がそれぞれ *xradius* と *yradius* で与えられる楕円弧を描き，その中をカレントフィルカラーとフィルパターンでフィルします。

戻り値　ありません。

可搬性　**fillellipse** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**arc**，**circle**，**ellipse**，**getaspectratio**，**pieslice**，**sector**，**setaspectratio**

例　**arc** を参照してください。

fillpoly

機能　多角形を描いてフィルします。

形式　# include <graphics.h>
void far fillpoly(int *numpoints*, int far * *polypoints*);

プロトタイプ　graphics.h

解説　**fillpoly** は，カレントラインスタイルとカラーで多角形を描き（**drawpoly** と同じ），カレントフィルスタイルとフィルカラーでその多角形をフィルします。

polypoints は，（*numpoints*×2）個の整数の並びを指します。整数の各ペアは，それぞれ多角形の頂点の x 座標と y 座標を与えます。

戻り値　ありません。

可搬性　**fillpoly** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**drawpoly**, **floodfill**, **graphresult**, **setfillstyle**

floodfill

機能 指定の境界カラーで囲まれた領域をフィルします。

形式
```
#include <graphics.h>
void far floodfill(int x, int y, int border);
```

プロトタイプ graphics.h

解説 **floodfill**は，ビットマップデバイス上の閉じた領域をフィルします。(*x*,*y*) は，閉じた領域内のシードポイント（フィルを開始する点）です。境界カラー ***border*** で囲まれた領域が，カレントフィルパターンとフィルカラーでフィルされます。シードポイントが閉じた領域の内側にある場合は領域の内側がフィルされ，外側にある場合は領域の外側がフィルされます。将来のバージョンとの互換性を保つためには，可能な限り **floodfill** ではなく **fillpoly** を用いてください。

注意：IBM-8514のドライバでは，**floodfill** は動作しません。

戻り値 領域をフィルしているときにエラーが発生した場合，graphresult は-7を返します。

可搬性 **floodfill** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **drawpoly**，**fillpoly**，**graphresult**，**setcolor**，**setfillstyle**

例

```
#include <graphics.h>

main()
{
   int graphdriver = DETECT, graphmode;  /* 自動検出を要求 */

   initgraph(&graphdriver, &graphmode, "");  /* 初期化 */

   /* 3Dバーを描いて，側面と上面をフィルする */
   setcolor(WHITE);
   setfillstyle(HATCH_FILL, LIGHTMAGENTA);
   bar3d(10, 10, 100, 100, 10, 1);
   setfillstyle(SOLID_FILL, LIGHTGREEN);
   floodfill(102, 50, WHITE);            /* 側面をフィル */
   floodfill(50, 8, WHITE);              /* 上面をフィル */

   closegraph();
}
```

getarccoords

機能　**arc** の最新の呼び出し時の座標を得ます。

形式　# include <graphics.h>

void far getarccoords(struct arccoordstype far * *arccoords*)；

プロトタイプ　graphics.h

解説　**getarccoords** は，*arccoords* が指す **arccoordstype** 構造体を，最後の **arc** の呼び出しに関する情報で埋めます。

arccoordstype 構造体は，graphics.h で次のように定義されています。

```
struct arccoordstype {
    int x, y;
    int xstart, ystart, xend, yend;
};
```

この構造体のメンバは，弧の中心（*x*,*y*），出発点（*xstart*,*ystart*），終了点（*xend*,*yend*）を示しています。これらの値は，弧の端を通る線を描く必要がある場合に便利です。

戻り値　ありません。

可搬性　**getarccoords** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**arc**，**fillellipse**，**pieslice**，**sector**

例　**arc** を参照してください。

getaspectratio

機能 カレントグラフィックモードにおけるアスペクト比を返します。

形式

```
#include <graphics.h>
void far getaspectratio(int far * xasp, int far * yasp);
```

プロトタイプ graphics.h

解説 *y* アスペクト因子＊ *yasp* は，10000に正規化されています。
PC-9801ではピクセルが正方形になっているので，＊ *xasp*＝＊ *yasp* となります。
IBM PC の場合，VGA を除くすべてのグラフィックスアダプタでは，ピクセルは横より縦が長いので，＊ *xasp* (*x* アスペクト因子) は＊ *yasp* より小さい値になります。VGA ではピクセルは正方形なので，＊ *xasp* と＊ *yasp* は等しくなります。一般に＊ *yasp* と＊ *xasp* の関係は次のようになっています。

```
* yasp =  10000
* xasp <= 10000
```

getaspectratio は，＊ *xasp* と＊ *yasp* の値を得ます。

戻り値 ありません。

可搬性 **getarccoords** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。Turbo Pascal に同様なルーチンがあります。

関連項目 **arc**, **circle**, **ellipse**, **fillellipse**, **pieslice**, **sector**, **setaspectratio**

例 **arc** を参照してください。

getbkcolor

機能　カレントバックグラウンドカラーを得ます。

形式　# include 〈graphics.h〉
int far getbkcolor(void)；

プロトタイプ　graphics.h

解説　**getbkcolor** は，カレントバックグラウンドカラーを返します(詳細については **setbkcolor** を参照してください)。

戻り値　**getbkcolor** はカレントバックグラウンドカラーを返します。

可搬性　**getbkcolor** は PC-9801シリーズ(または IBM PC とその互換機)でのみ動作します(IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**getcolor**，**getmaxcolor**，**getpalette**，**setbkcolor**

例

```
#include <graphics.h>
#include <conio.h>
#include <dos.h>

main()
{
   int graphdriver = DETECT, graphmode;    /* 自動検出を要求 */
   int svcolor;

   initgraph(&graphdriver, &graphmode, "");        /* 初期化 */

   svcolor = getbkcolor();  /* カレントバックカラーをセーブ */
   setbkcolor(sccolor ^ 1); /* カレントバックカラーを変更   */
   delay(5000);             /* 5 秒間待つ                   */
   setbkcolor(svcolor);     /* 元のカラーに戻す             */

   getche();
   closegraph();
}
```

getcolor

機能 カレントドロウカラーを得ます。

形式 # include <graphics.h>
int far getcolor(void);

プロトタイプ graphics.h

解説 **getcolor** は，カレントドロウカラーを返します。
その範囲は0から **getmaxcolor**() までです。ドロウカラーとは，線などを描くときにピクセルにセットされる値のことです。たとえば PC98C8モードでは，**getcolor**() が4を返せばカレントドロウカラーは紫色になります。

戻り値 **getcolor** はカレントドロウカラーを返します。

可搬性 **getcolor** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **getbkcolor**, **getmaxcolor**, **getpalette**, **setcolor**

例

```
#include <graphics.h>
#include <conio.h>

main()
{
   int graphdriver = DETECT, graphmode;   /* 自動検出を要求 */
   int svcolor;

   initgraph(&graphdriver, &graphmode, "");   /* 初期化 */

   svcolor = getcolor();   /* カレントドロウカラーをセーブ */
   setcolor(3);            /* ドロウカラーにパレット番号3を設定 */
   circle(100, 100, 5);    /* カラーの小さな円 */
   setcolor(svcolor);      /* ドロウカラーを元に戻す */

   getche();
   closegraph();
}
```

getdefaultpalette

機能　パレット定義構造体を返します。

形式　# include <graphics.h>

```
void far * far getdefaultpalette(void);
```

プロトタイプ　graphics.h

解説　**getdefaultpalette** は，**initgraph** の呼び出し中にドライバによって初期化されたパレットを持っている **palettetype** 構造体を見つけます。

戻り値　**getdefaultpalette** は，カレントドライバが初期化されたときに，そのドライバによってセットアップされたデフォルトパレットへのポインタを返します。

可搬性　**getdefaultpalette** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getpalette**，**initgraph**

getdrivername

機能　カレントグラフィックスドライバの名前を持つ文字列を返します。

形式　# include <graphics.h>

```
char * far getdrivername(void);
```

プロトタイプ　graphics.h

解説　**initgraph** を呼び出し後，**getdrivername** は現在ロードされているグラフィックスドライバの名前を返します。

戻り値　**getdrivername** は，グラフィックスドライバを示す文字列へのポインタを返します。

可搬性　**getdrivername** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getmodename**，**initgraph**

getfillpattern

機能　ユーザ定義のフィルパターンをメモリにコピーします。

形式　# include <graphics.h>
void far getfillpattern(char far * *pattern*);

プロトタイプ　graphics.h

解説　**getfillpattern** は, **setfillpattern** で設定されたユーザ定義のフィルパターンを, *pattern* が指している8バイト領域にコピーします。
pattern は8バイト列を指すポインタで, 各バイトはパターン内の8ピクセルに対応しています。パターン内で1に設定されているビットに対応するピクセルがプロットされます。たとえば, 次のユーザ定義のフィルパターンはチェッカー盤を表わしています。

```
char checkerboard[8] = {
    0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55
};
```

戻り値　ありません。

可搬性　**getfillpattern** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では, サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連事項　**getfillsettings**, **setfillpattern**

getfillsettings

機能 カレントフィルパターンとフィルカラーに関する情報を得ます。

形式 # include <graphicis.h>

void far getfillsettings(struct fillsettingstype far * *fillinfo*);

プロトタイプ graphics.h

解説 **getfillsettings** は，*fillinfo* が指している **fillsettingstype** 型構造体に，カレントフィルパターンとフィルカラーを書き込みます。**fillsettingstype** は，graphics.h の中で次のように定義されています。

```
struct fillsettingstype {
    int pattern;          /* カレントフィルパターン */
    int color;            /* カレントフィルカラー   */
};
```

bar，**bar3d**，**fillpoly**，**floodfill**，**pieslice** は，すべてカレントフィルパターンとフィルカラーで領域をフィルします。定義済みのフィルパターンは11種類あります（ベタ塗りやクロスハッチ，ドットなど)。

定義済みパターンのシンボル名は，graphics.h の *fill_patterns* テーブルに与えられています(下の表を参照)。これに加えてユーザは独自のフィルパターンを定義することができます。

pattern が12（USER_FILL）に等しい場合は，ユーザ定義のフィルパターンが用いられています。それ以外の場合は定義済みパターンの数値が与えられます。

graphics.h で定義されている列挙型 *fill_pattern* は，定義済みフィルパターンの名前とユーザ定義パターンの指示子を与えています。

名前	値	機能説明
EMPTY_FILL	0	バックグラウンドカラーでフィル(中空)
SOLID_FILL	1	ベタ塗り
LINE_FILL	2	横線でフィル
LTSLASH_FILL	3	///でフィル
SLASH_FILL	4	///でフィル，太線
BKSLASH_FILL	5	＼＼＼でフィル，太線
LTBKSLASH_FILL	6	＼＼＼でフィル
HATCH_FILL	7	淡いハッチ（格子）でフィル
XHATCH_FILL	8	濃いクロスハッチ(斜めの格子)でフィル
INTERLEAVE_FILL	9	インターリーブ線でフィル
WIDE_DOT_FILL	10	すき間が広い点でフィル
CLOSE_DOT_FILL	11	すき間が狭い点でフィル
USER_FILL	12	ユーザ定義のフィルパターン

EMPTY_FILL 以外のものは，カレントフィルカラーでフィルします。EMPTY_FILL は，カレントバックグラウンドカラーを用います。

戻り値　　ありません。

可搬性　　**getfillsettings** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連事項　　**getfillpattern**，**setfillpattern**，**setfillstyle**

例

```
#include <graphics.h>
#include <conio.h>

main()
{
   int graphdriver = DETECT, graphmode; /* 自動検出を要求 */
   struct fillsettingstype save;
   char savepattern[8];
   char gray50[] =
       { 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55 };

   initgraph(&graphdriver, &graphmode, "");   /* 初期化 */

   getfillsettings(&save);          /* 現在の設定を得る */
   if (save.pattern == USER_FILL)  /* ユーザ定義パターンならば */
      getfillpattern(savepattern); /* そのパターンをセーブ */

   setfillstyle(SLASH_FILL, BLUE); /* フィルスタイルを変更 */
   bar(  0,   0, 100, 100);         /* 青で斜線フィルのバーを描く */

   setfillpattern(gray50, YELLOW); /* ユーザ定義パターンで */
   bar(100, 100, 200, 200);         /* 黄色のバーを描く */

   if (save.pattern == USER_FILL)  /* ユーザ定義パターンならば */
                                    /* ユーザ定義パターンを回復 */
      setfillpattern(savepattern, save.color);
   else                             /* 元のスタイルを回復 */
      setfillstyle(save.pattern, save.color);

   getche();
   closegraph();
}
```

getgraphmode

機能　カレントグラフィックスモードを返します。

形式　#include <graphics.h>
int far getgraphmode(void);

プロトタイプ　graphics.h

解説　**getgraphmode** を呼び出す前に，プログラムは **initgraph** の呼び出しに成功していなければなりません。
graphics.h で定義されている列挙型 *graphics_mode* は，システム定義のグラフィックスモードの名前を与えています。この列挙型の値については **initgraph** の解説を参照してください。

戻り値　**getgraphmode** は，**initgraph** または **setgraphmode** で設定されたカレントグラフィックスモードを返します。

可搬性　**getgraphmode** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getmoderange**，**estorecrtmode**，**setgraphmode**

例

```
int cmode;

cmode = getgraphmode();      /* カレントモードをセーブ */
restorecrtmode();            /* テキストモードに移行 */
printf("Now in text mode - "
       "press any key to go back to graphics ...");

getch();
setgraphmode(cmode);         /* グラフィックスモードに戻る */
```

getimage

機能　指定された領域のビットイメージをメモリへセーブします。

形式　# include <graphics.h>
void far getimage(int *left*, int *top*, int *right*,
　　　　int *bottom*, void far * *bitmap*);

プロトタイプ　graphics.h

解説　**getimage** は，画面上の長方形領域のビットイメージをメモリにコピーします。*left*, *top*, *right*, *bottom* は，コピーされる画面上の長方形を定義します。*bitmap* は，ビットイメージが格納されるメモリ内の領域を指します。この領域の最初の2ワードには長方形の幅と高さが格納され，残りの部分にイメージそのものが格納されます。

戻り値　ありません。

可搬性　**getimage** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**imagesize**, **putimage**, **putpixel**

例

```
#include <alloc.h>
#include <graphics.h>

main()
{
   int graphdriver = DETECT, graphmode;    /* 自動検出を要求 */
    void *buffer;
    unsigned size;

    initgraph(&graphdriver, &graphmode, "")    /* 初期化 */

    size = imagesize(0, 0, 20, 10);
    buffer = malloc(size);              /* イメージ用のメモリを確保 */
    getimage(0, 0, 20, 10, buffer);     /* イメージを取り込む */
    /* ...  */
    putimage(0, 0, buffer, COPY_PUT);  /* イメージを復元 */
    free(buffer);                       /* buffer を解放 */

    closegraph();
}
```

getlinesettings

機能　カレントラインスタイル，パターン，幅を得ます。

形式　#include <graphics.h>

void far getlinesettings(struct linesettingstype far * *lineinfo*) ;

プロトタイプ　graphics.h

解説　**getlinesettings** は，*lineinfo* が指している **linesettingstype** 構造体に，カレントラインスタイル，パターン，幅に関する情報を書き込みます。

linesettingstype 構造体は，graphics.h で次のように定義されています。

```
struct linesettingstype {
    int linestyle;
    unsigned upattern;
    int thickness;
};
```

linestyle は，それ以降に描かれるラインのスタイルを指定します（たとえば，実線，点線，一点鎖線，破線）。graphics.h で定義されている *line_style* テーブルには，これらの演算子の名前が与えられています。

名前	値	意味
SOLID_LINE	0	実線
DOTTED_LINE	1	点線
CENTER_LINE	2	一点鎖線
DASHED_LINE	3	破線
USERBIT_LINE	4	ユーザ定義のラインスタイル

thickness は，それ以降に描かれる線の幅が，普通か太いかを指定するものです。

名前	値	意味
NORM_WIDTH	1	1ピクセル幅
THICK_WIDTH	3	3ピクセル幅

upattern は，*linestyle* が USERBIT_LINE(4)のときにのみ適用される16ビットパターンです。その場合，パターンワード内の1のビットに対応するライン内のピクセルが，カレントドロウカラーで描かれます。たとえば，実線は 0xFFFF の upattern（すべてのピクセルの描画）に対し、破線は 0x3333または0x0F0F の *upattern* に対応します。**setlinestyle** の引数 *linestyle* が USERBIT_LINE(4)でない場合も，*upattern* は与えなければなりません（無視されますが)。

戻り値　　ありません。

可搬性　　**getlinesettings** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　　**setlinestyle**

例

```
#include <graphics.h>
#include <conio.h>

main()
{
   int graphdriver = DETECT, graphmode;      /* 自動検出を要求 */
   struct linesettingstype saveline;

   initgraph(&graphdriver, &graphmode, "");      /* 初期化 */

   getlinesettings(&saveline);  /* ラインスタイルをセーブ */
   setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
   rectangle(10, 10, 17, 15);    /* 太いラインの小さなボックス */
   setlinestyle(saveline.linestyle,    /* 元のライン設定を回復 */
                saveline.upattern,
                saveline.thickness);
   getche();
   closegraph();
}
```

getmaxcolor

機能　**setcolor** 関数に渡すことができるカラーの最大値を返します。

形式　# include <graphics.h>
int far getmaxcolor(void)；

プロトタイプ　graphics.h

解説　**getmaxcolor** は，カレントグラフィックスドライバと，カレントグラフィックスモードにおいて有効なカラーの最大値（パレットサイズ-1）を返します。
PC-9801では，**getmaxcolor** は常に15を返し，これは **setcolor** に渡す数値は0から15までが有効であることを意味します。
IBM PC の場合，たとえば256K の EGA では，**getmaxcolor** は常に15を返し，これは **setcolor** には0から15までの数値を指定できることを意味します。また，CGA の高解像度モードや Hercules モノクロアダプタでは，0および1のカラーしかサポートしていないため，**getmaxcolor** は常に1を返します。

戻り値　**getmaxcolor** は使用できるカラーの最大値を返します。

可搬性　**getmaxcolor** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getbkcolor**，**getcolor**，**getpalette**，**getpalettesize**，**setcolor**

getmaxmode

機能　　カレントドライバで使用できる最大のモード番号を返します。

形式　　# include <graphics.h>

int far getmaxmode(void)；

プロトタイプ　graphics.h

解説　　**getmaxmode**は，現在ロードされているドライバで使用可能な最大のモード番号を，そのドライバから直接取り出して返します。**getmaxmode**は，サードパーティが提供するドライバに対しても機能します（**getmoderange**はデフォルトのドライバに対してしか機能しません）。モードの最小値は0です。

戻り値　　**getmaxmode**は，カレントドライバにおいて使用可能なモードの最大値を返します。

可搬性　　**getmaxmode**はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　　**getmodename**，**getmoderange**

getmaxx

機能　画面上の x 座標の最大値を返します。

形式　# include <graphics.h>
int far getmaxx(void);

プロトタイプ　graphics.h

解説　**getmaxx** は，カレントグラフィックスドライバと，カレントグラフィックスモードで有効な，x の最大値（画面に対する相対値）を返します。
PC-9801では，**getmaxx** は常に639を返します。
IBM PC の場合，たとえば CGA の320×200モードでは，**getmaxx** は319を返します。
getmaxx は，センタリングや画面上の領域の境界を決定する際に有用です。

戻り値　**getmaxx** は，画面座標における x 座標の最大値を返します。

可搬性　**getmaxx** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getmaxy**，**getx**

例

```
printf("The screen resolution is %d pixels by %d pixels.¥n",
       getmaxx()+1, getmaxy()+1);
```

getmaxy

機能　画面上の *y* 座標の最大値を返します。

形式
```
#include <graphics.h>
int far getmaxy(void);
```

プロトタイプ　graphics.h

解説　**getmaxy** は，カレントグラフィックスドライバと，カレントグラフィックスモードで有効な，*y* の最大値（画面に対する相対値）を返します。

PC-9801では，**getmaxy** は常に399を返します。

IBM PC の場合，たとえば CGA の320×200モードでは，**getmaxx** は199を返します。

getmaxy は，センタリングや画面上の領域の境界を決定する際に有用です。

戻り値　**getmaxy** は画面座標における *y* 座標の最大値を返します。

可搬性　**getmaxy** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getmaxx**, **gety**

例　**getmaxx** を参照してください。

getmodename

機能 グラフィックスモード名を得ます。

形式 # include <graphics.h>
char ＊ far getmodename(int *mode_number)*；

プロトタイプ graphics.h

解説 **getmodename** はカレントグラフィックスドライバの *mode_number* によって指定されたモードの名前を示す文字列へのポインタを返します。モード名はドライバごとに組み込まれています。返される値は，メニューを作成したりステータスを表示したりする際に便利です。
PC-9801では以下のような文字列が返されます。

mode_number	**文字列**
0	"640 x 400 PC9801 8_COLOR"
1	"640 x 400 PC9801 16_COLOR"

戻り値 **getmodename** はモード名へのポインタを返します。

可搬性 **getmodename** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **getdrivername**，**getmaxmode**，**getmoderange**

getmoderange

機能　指定のグラフィックスドライバで有効なモードの範囲を得ます。

形式　# include <graphics.h>

void far getmoderange(int *graphdriver*, int far * *lomode*,
　　　　　　　　　　int far * *himode*);

プロトタイプ　graphics.h

解説　**getmoderange** は，与えられたグラフィックスドライバ *graphdriver* において有効なグラフィックスモードの範囲を得ます。許されるモードの最小値は * *lomode* に返され，最大値は * *himode* に返されます。*graphdriver* として無効なグラフィックスドライバが指定されると，* *lomode* と * *himode* はともに-1にセットされます。* *graphdriver* の値が-1の場合には，現在ロードされているドライバを指定することになります。

戻り値　ありません。

可搬性　**getmoderange** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getgraphmode, getmaxmode, getmodename, initgraph, setgraphmode**

例

```
#include <graphics.h>

main()
{
   int lo, hi;

   getmoderange(PC98, &lo, &hi);
   printf("PC-9801 supports modes %d through %d¥n", lo, hi);
}
```

getpalette

機能 カレントパレットに関する情報を得ます。

形式 # include <graphics.h>

void far getpalette(struct palettetype far * *palette*) ;

プロトタイプ graphics.h

解説 **getpalette** は，カレントパレットのサイズと各カラーに関する情報を，*palette* が指す **palettetype** 型の構造体に書き込みます。

getpalette で用いられる **palettetype** 型構造体と定数 MAXCOLORS は，graphics.h の中で次のように定義されています。

```
#define MAXCOLORS  15

struct palettetype {
    unsigned char size;
    signed char colors[MAXCOLORS + 1];
};
```

size は，カレントグラフィックスドライバと，カレントグラフィックスモードでの，パレット中のカラーの数です。

colors は，パレットの中のそれぞれの項目に対応した実際のカラーの番号を要素とする *size* バイトの配列です。

注意：**getpalette** は，IBM-8514ドライバでは使用できません。

戻り値 ありません。

可搬性 **getpalette** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **getbkcolor**，**getcolor**，**getdefaultpalette**，**getmaxcolor**，**setallpalette**，**setpalette**

例

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>

main()
{
  int graphdriver = DETECT, graphmode;      /* 自動検出を要求 */
  struct palettetype palette;
  int color;

  initgraph(&graphdriver, &graphmode, "");  /* 初期化 */

  getpalette(&palette);                     /* カレントパレットを得る */
  for(color = 0; color < palette.size; color++)
  {
    setfillstyle(SOLID_FILL, color);  /* 様々な色のバーを描く */
    bar(20*(color-1), 0, 20*color, 20);
  }

  if (palette.size > 1) {                   /* カラーが複数あれば */
    do                                      /* キーが押されるまで */
      setpalette(random(palette.size), random(palette.size));
    while(!kbhit());                        /* ランダムに切り換える */
    getch();                                /* 入力された文字は捨てる */
  }

  setallpalette(&palette);                  /* パレットを元に戻す */

  closegraph();
}
```

getpalettesize

機能　パレットカラー参照テーブルのサイズを返します。

形式
```
#include <graphics.h>
int far getpalettesize(void);
```

プロトタイプ　graphics.h

解説　**getpalettesize** は，カレントグラフィックスモードで，いくつのパレットが設定できるかを得るために使用します。
PC-9801では，**getpalettesize** は常に16を返します。
IBM PC の場合，たとえば EGA のカラーモードでは16が返されます。

戻り値　**getpalettesize** は，カレントパレットの項目数を返します。

可搬性　**getpalettesize** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**setpalette**，**setallpalette**

getpixel

機能 指定されたピクセルのカラーを得ます。

形式 # include <graphics.h>
unsigned far getpixel(int *x*, int *y*) ;

プロトタイプ graphics.h

解説 **getpixel** を使うと，(*x*,*y*)に位置するピクセルのカラーが得られます。

戻り値 **getpixel** は指定されたピクセルのカラーを返します。

可搬性 **getpixel** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目 **getimage**, **putpixel**

例

```
#include <graphics.h>
#include <conio.h>

main()
{
   int graphdriver = DETECT, graphmode;      /* 自動検出を要求 */
   int i, color, max;

   initgraph(&graphdriver, &graphmode, "");   /* 初期化 */
   max = getmaxcolor() + 1;

   /* ライン上のピクセルのカラーを変化させる */
   for (i = 1; i < 200; i++) {
      color = getpixel(i, i);
      putpixel(i, i, (color ^ i) % max);
   }
   getche();
   closegraph();
}
```

gettextsettings

機能　現在のグラフィックテキストの設定に関する情報を返します。

形式　# include <graphics.h>
void far gettextsettings(struct textsettingstype far * *texttypeinfo*)；

プロトタイプ　graphics.h

解説　**gettextsettings** は，*texttypeinfo* が指す **textsettingstype** 型構造体に，カレントテキストフォント，テキストの向き，文字サイズ，位置合わせについての情報（**settextstyle** や **settextjustify** でセットされる情報）を書き込みます。

gettextsettings で用いられる **textsettingstype** 型構造体は，graphics.h の中で次のように定義されています。

```
struct textsettingstype {
    int font;
    int direction;
    int charsize;
    int horiz;
    int vert;
}
```

各フィールドの意味については **settextstyle** を参照してください。

戻り値　ありません。

可搬性　**gettextsettings** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**outtext**，**outtextxy**，**settextjustify**，**settextstyle**，**setusercharsize**，**textheight**，**textwidth**

例

```
#include <graphics.h>
#include <conio.h>

main()
{
   int graphdriver = DETECT, graphmode;    /* 自動検出を要求 */
   struct textsettingstype oldtext;

   initgraph(&graphdriver, &graphmode,""); /* 初期化 */

   gettextsettings(&oldtext);              /* 現在の設定を得る */

   /* 水平方向，上左揃えに切り替え
      ゴシックフォント，因子5       */
   settextjustify(LEFT_TEXT, TOP_TEXT);
   settextstyle(GOTHIC_FONT, HORIZ_DIR, 5);
   outtext("Gothic Text");

   /* 設定を元に戻す */
   settextjustify(oldtext.horiz, oldtext.vert);
   settextstyle(oldtext.font, oldtext.direction,
                oldtext.charsize);
   getche();
   closegraph();
}
```

getviewsettings

機能　カレントビューポートに関する情報を得ます。

形式　# include <graphics.h>
void far getviewsettings(struct viewporttype far ＊ *viewport*)；

プロトタイプ　graphics.h

解説　**getviewsettings** は，*viewport* が指す **viewporttype** 型構造体に，カレントビューポートに関する情報を書き込みます。
getviewport が用いる **viewpoint** 型構造体は，graphics.h の中で次のように定義されています。

```
struct viewporttype {
    int left, top, right, bottom;
    int clipflag;
}
```

戻り値　ありません。

可搬性　**getviewsettings** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**clearviewport**，**getx**，**gety**，**setviewport**

例

```
struct viewporttype view;

getviewsettings(&view);         /* 現在の設定を得る */
if (!view.clip)                 /* クリッピングがオフであれば */
   setviewport(view.left, view.top,          /* オンにする */
               view.right, view.bottom, 1);
```

getx

機能　現在位置の x 座標を返します。

形式

```
#include <graphics.h>
int far getx(void);
```

プロトタイプ　graphics.h

解説　**getx** は，現在位置の x 座標を返します。返される値はビューポート相対です。

戻り値　**getx** は，CP の x 座標を返します。

可搬性　**getx** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getmaxx**，**getmaxy**，**getviewsettings**，**gety**

例

```
int oldx, oldy;

/* カレント位置をセーブ */
oldx = getx();
oldy = gety();

circle(100, 100, 2);      /* [100,100]に小さな丸を描く */
moveto(99, 100);
linerel(2, 0);
moveto(oldx, oldy);       /* 元の位置に戻る */
```

gety

機能　　現在位置の y 座標を返します。

形式

```
#include <graphics.h>
int far gety(void);
```

プロトタイプ　graphics.h

解説　　**gety** は，現在位置の y 座標を返します。返される値はビューポート相対です。

戻り値　**gety** は，CP の y 座標を返します。

可搬性　**gety** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getmaxx**，**getmaxy**，**getviewsettings**，**getx**

例　　**getx** を参照してください。

graphdefaults

機能　グラフィックスのすべての設定をデフォルトにリセットします。

形式
```
#include &lt;graphics.h&gt;
void far graphdefaults(void);
```

プロトタイプ　graphics.h

解説　**graphdefaults**は，あらゆるグラフィックスの設定をデフォルトにリセットします。これは以下の動作を意味します。

■ビューポートをスクリーン全体に設定する。
■カレント座標を（0,0）に動かす。
■デフォルトのパレットカラー，バックグランドカラー，ドロウカラーをセットする。
■デフォルトのフィルスタイルとフィルパターンをセットする。
■デフォルトのテキストフォントと位置合わせをセットする。

戻り値　ありません。

可搬性　**graphdefaults**はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**initgraph**

grapherrormsg

機能 エラーメッセージ文字列へのポインタを返します。

形式

```
# include <graphics.h>
char * far grapherrormsg(int errorcode);
```

プロトタイプ graphics.h

解説 **grapherrormsg** は，**graphresult** が返したエラーコードに対応するエラーメッセージ文字列を指すポインタを返します。
errorcode の値と対応する文字列については，**graphresult** の解説を参照してください。

戻り値 **grapherrormsg** は，エラーメッセージ文字列を指すポインタを返します。

可搬性 **grapherrormsg** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **graphresult**

_graphfreemem

機能 グラフィックスメモリの解放ルーチンへのフックです。

形式

```
#include <graphics.h>
void far _graphfreemem(void far * ptr, unsigned size);
```

プロトタイプ graphics.h

解説 グラフィックスライブラリ中のルーチンは，**_graphfreemem** を呼び出して，前に **_graphgetmem** を使って割り当てたメモリを解放します。ユーザ独自の **_graphfreemem** のバージョンを定義すれば（これは上の宣言通りに正確に宣言しなければいけません），グラフィックスライブラリのメモリ管理の方法を変更することができます。デフォルトバージョンでは，このルーチンは **free** を呼び出すだけです。

戻り値 ありません。

可搬性 **_graphfreemem** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **_graphgetmem**, **setgraphbufsize**

例

```
/* ユーザ定義のグラフィックスメモリ管理ルーチンの例 */

#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>

main()
{
   int errorcode;
   int graphdriver;
   int graphmode;

   graphdriver = DETECT;
   initgraph(&graphdriver, &graphmode, "a:¥¥");
   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("graphics error: %s¥n", grapherrormsg(errorcode));
      exit(1);
   }

   settextstyle(GOTHIC_FONT, HORIZ_DIR, 4);
   outtextxy(100, 100, "BGI TEST");
   getche();
   closegraph();
}

void far *far _graphgetmem(unsigned size)
{
   printf("_graphgetmem called [size=%d] -- hit any key", size);
   getch(); printf("¥n");
   return(farmalloc(size));                /* farヒープを使用 */
}

void far _graphfreemem(void far *ptr, unsigned size)
{
   printf("_graphfreemem called [size=%d] -- hit any key", size);
   getch(); printf("¥n);
   farfree(ptr);                           /* sizeは使われない */
}
```

_graphgetmem

機能　　グラフィックスメモリの割り当てルーチンへのフックです。

形式

```
#include <graphics.h>
void far * far _graphgetmem(unsigned size);
```

プロトタイプ　graphics.h

解説　　グラフィックスライブラリ中のルーチンは，通常**_graphgetmem**を呼び出して，内部バッファ，グラフィックスドライバ，文字セットのためのメモリを割り当てます(**_graphgetmem**を呼び出すのはライブラリルーチンであり，ユーザプログラムではありません)。ユーザ独自の**_graphgetmem**を定義すれば（形式に示されているとおりに正確に宣言しなければいけませんが），グラフィックスライブラリのメモリ管理の方法を変更することができます。デフォルトでは，このルーチンは**malloc**を呼び出すだけです。

戻り値　　ありません。

可搬性　　**_graphgetmem**はPC-9801シリーズ(またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　　**_graphfreemem**，**initgraph**，**setgraphbufsize**

例　　**_graphfreemem**を参照してください。

graphresult

機能　失敗した最後のグラフィックス操作に対するエラーコードを返します。

形式

```
# include <graphics.h>
int far graphresult(void) ;
```

プロトタイプ　graphics.h

解説　**graphresult** は，エラーとなった最後のグラフィックス操作に対するエラーコードを返し，エラーレベルを grOk にリセットします。
次の表は，**graphresult** が返すエラーコードを示しています。この表の中の定数は，graphics.h の中で列挙型 *graphics_errors* として定義されています。

エラーコード	*graphics_errors* **定数**	**対応するエラーメッセージ文字列 / 意味**
0	grOk	No error エラーなし
-1	grNoInitGraph	(BGI) graphics not installed (use **initgraph**) (BGI) グラフィックスが組み込まれていない（**initgraph** を使用せよ）
-2	grNotDetected	Graphics hardware not detected グラフィックスハードウェアが検出できない
-3	grFileNotFound	Device driver file not found デバイスドライバファイルがみつからない

-4	grInvalidDriver	Invalid device driver file デバイスドライバファイルが正しくない
-5	grNoLoadMem	Not enough memory to load driver ドライバをロードするメモリが不足
-6	grNoScanMem	Out of memory in scan fill スキャンフィルでメモリが不足
-7	grNoFloodMem	Out of memory in flood fill フラッドフィルでメモリが不足
-8	grFontNotFound	Font file not found フォントファイルがみつからない
-9	grNoFontMem	Not enough memory to load font フォントをロードするメモリが不足
-10	grInvalidMode	Invalid graphics mode for selected driver 選んだドライバに対してグラフィックスモードが正しくない
-11	grError	Graphics error グラフィックスエラー
-12	grIOerror	Graphics I/O error グラフィックスI/Oエラー
-13	grInvalidFont	Invalid font file フォントファイルが正しくない
-14	grInvalidFontNum	Invalid font number フォント番号が正しくない
-15	grInvalidDeviceNum	Invalid device number デバイス番号が正しくない
-18	grInvalidVersionNum	Invalid device number デバイス番号が正しくない

graphresult が呼び出されると，エラーコードは0にリセットされることに注意してください。したがって，**graphresult** が返した値をいったん変数に代入してからテストすべきです。

戻り値

graphresult は，現在のグラフィックスエラー番号，すなわち-18～0の範囲の整数を返します。**grapherrormsg** を使うと，**graphresult** が返したエラーコードに対応するエラーメッセージ文字列を指すポインタを得ることができます。

可搬性

graphresult は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目

detectgraph, drawpoly, fillpoly, floodfill, grapherrormsg, initgraph, pieslice, registerbgidriver, registerbgifont, setallpalette, setcolor, setfillstyle, setgraphmode, setlinestyle, setpalette, settextjustify, settextstyle, setusercharsize, setviewport, setvisualpage

imagesize

機能 ビットイメージの格納に必要なバイト数を返します。

形式 # include <graphics.h>
unsigned far imagesize(int *left*, int *top*, int *right*, int *bottom*) ;

プロトタイプ graphics.h

解説 **imagesize** は，**getimage** によってビットイメージを格納するのに必要なメモリ領域のサイズを決定します。返されるバイト数には，イメージの幅と高さのためのメモリ領域も含まれています。選んだイメージに対して必要となる大きさが64K-1バイト以上の場合，**imagesize** は0xFFFF（-1）を返します。

戻り値 **imagesize** は，必要とされるメモリ領域のサイズをバイト単位で返します。

可搬性 **imgesize** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **getimage**，**putimage**

initgraph

機能　グラフィックスシステムを初期化します。

形式　# include <graphics.h>
void far initgraph(int far * ***graphdriver***, int far * ***graphmode***,
　　　　char far * ***pathtodriver***);

プロトタイプ　graphics.h

解説　**initgraph** は，ディスクからグラフィックスドライバをロード(もしくは登録されているドライバを有効に）し，システムをグラフィックスモードにして，グラフィックスシステムを初期化します。

グラフィックスシステムを起動するには，まず最初に **initgraph** を呼び出します。**initgraph** は，グラフィックスドライバをロードし，システムをグラフィックスモードにします。**initgraph** には，特定のグラフィックスドライバおよびモードを指定することができます。あるいは，実行時にハードウェアを調べて自動的に最適なドライバをロードするように指示すること（自動検出）もできます。

自動検出を指示すると，**initgraph** はグラフィックスドライバとモードを選択するために detectgraph を呼び出します。**initgraph** はさらに，すべてのグラフィックスの設定（現在位置，パレット，カラー，ビューポートなど）をデフォルトにセットし，**graphresult** を0にリセットします。

通常 **initgraph** は（**_graphgetmem** を使って）ドライバに必要なメモリを割り当て，ディスクから.BGI ファイルを読み込むことによって，グラフィックスドライバをロードします。このように動的にロードする方法の他に，グラフィックスドライバファイル（場合によってはそのいくつか）を実行可能プログラムファイルに直接リンクする方法があります（この詳細についてはユーザーズガイド付録Cの BGIOBJ の節を参照してください)。

pathtodriver には，**initgraph** がグラフィックスドライバを探索するディレクトリパスを指定します。**initgraph** は，まず *pathtodriver* で指定されたパスで探索を行ない，（グラフィックスドライバがそこになければ）次にカレントディレクトリで探索を行ないます。したがって，*pathtodriver* が NULL であると，ドライバファイル（*.BGI）はカレントディレクトリになければなりません。このパスは，**settextstyle** がストローク文字フォントファイル（*.CHR）を探索するパスでもあります。

* *graphdriver* は，使用されるべきグラフィックスドライバを指定する整数です。この値には，graphics.h に定義されている次の表に示す列挙型 *graphics_drivers* 中の定数を使うことができます。

graphics_drivers **定数**	**数値**
DETECT	0（自動検出要求）
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10
PC98	11（PC-9801のみ）
PC98GRCG	12（PC-9801のみ）
PC98EGC	13（PC-9801のみ）

* *graphmode* は，初期グラフィックスモードを示す整数です（ただし，* *graphdriver* が DETECT に等しい場合には，* *graphmode* は **initgraph** によって，検出されたドライバにおいて最も高い解像度が得られるモードにセットされます)。* *graphmode* には，graphics.h で定義されている以下のような列挙型 *graphics_modes* の定数を与えることができます。

ドライバ	*graphics_modes*	値	横×縦	パレット	ページ
PC98	PC98C8	0	640×400	8色	2
	PC98C16	1	640×400	16色	2
PC98GRCG	PC98C8	0	640×400	8色	2
	PC98C16	1	640×400	16色	2
PC98EGC	PC98C8	0	640×400	8色	2
	PC98C16	1	640×400	16色	2
CGA	CGAC0	0	320×200	C0	1
	CGAC1	1	320×200	C1	1
	CGAC2	2	320×200	C2	1
	CGAC3	3	320×200	C3	1
	CGAHI	4	640×200	2色	1
MCGA	MCGAC0	0	320×200	C0	1
	MCGAC1	1	320×200	C1	1
	MCGAC2	2	320×200	C2	1
	MCGAC3	3	320×200	C3	1
	MCGAMED	4	640×200	2色	1
	MCGAHI	5	640×480	2色	1
EGA	EGALO	0	640×200	16色	4
	EGAHI	1	640×350	16色	2

EGA64	EGA64LO	0	640×200	16色	1
	EGA64HI	1	640×350	4色	1
EGAMONO	EGAMONOHI	3	640×350	2色	1*
	EGAMONOHI	3	640×350	2色	2**
HERC	HERCMONOHI	0	720×348	2色	2
ATT400	ATT400C0	0	320×200	C0	1
	ATT400C1	1	320×200	C1	1
	ATT400C2	2	320×200	C2	1
	ATT400C3	3	320×200	C3	1
	ATT400MED	4	640×200	2色	1
	ATT400HI	5	640×400	2色	1
VGA	VGALO	0	640×200	16色	2
	VGAMED	1	640×350	16色	2
	VGAHI	2	640×480	16色	1
PC3270	PC3270	0	720×350	2色	1
IBM8514	IBM8514HI	0	640×480	256色	2
	IBM8514LO	1	1024×768	256色	2

*　64K EGA モノカード

** 256K EGA モノカード

注意：* *graphdriver* と * *graphmode* には，上の表の中から有効な値を指定しなければなりません。そうしないと期待通りの結果が得られないことがあります。唯一の例外は * *graphdriver*＝DETECT の場合です。

上の表中のC0，C1，C2およびC3は，CGA（およびその互換）システムで使用できる4つの定義済みの4色のパレットを参照しています。これらのパレットではバックグラウンドカラー（項目#1）のみが指定可能で，他のカラーは固定です。この点の詳細については，ユーザーズガイド第8章の「**IBM PC CGA でのカラー制御**」の節で述べられていますが，次の表にまとめだけを示します。

	カラー番号（ピクセル値）に割り当てられている定数		
パレット番号	**1**	**2**	**3**
0	CGA_LIGHTGREEN	CGA_LIGHTRED	CGA_YELLOW
1	CGA_LIGHTCYAN	CGA_LIGHTMAGENTA	CGA_WHITE
2	CGA_GREEN	CGA_RED	CGA_BROWN
3	CGA_CYAN	CGA_MAGENTA	CGA_LIGHTGRAY

initgraph を呼び出すと，* *graphdriver* はカレントグラフィックスドライバに，* *graphmode* はカレントグラフィックスモードにセットされます。

戻り値

initgraph は，常に内部エラーコードをセットします。成功した場合，エラーコードには0がセットされます。エラーが発生した場合は，* *graphdriver* は-2，-3，-4，あるいは-5にセットされ，**graphresult** も同じ値を返します。各値は次のような意味を持ちます。

-2	グラフィックスカードが検出できない
-3	ドライバファイルを検出できない
-4	ドライバが正しくない
-5	ドライバをロードするためのメモリが足りない

可搬性

initgraph は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　closegraph, detectgraph, getdefaultpalette, getdrivername, getmoderange, graphdefaults, _graphgetmem, graphresult, installuserdriver, registerbgidriver, registerbgifont, restorecrtmode, setgraphbufsize, setgraphmode

例

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>

main()
{
   int g_driver, g_mode, g_error;
   detectgraph(&g_driver, &g_mode);
   if (g_driver < 0)
   {
      printf("No graphics hardware detected !¥n);
      exit(1);
   }

   printf("Detected graphics driver #%d, mode #%d¥n",
          g_driver, g_mode);
   getch();
   if (g_mode == EGAHI)
      /* override mode if EGA detected */
      g_mode = EGALO;
   initgraph(&g_driver, &g_mode, "");
   g_error = graphresult();

   if (g_error < 0)
   {
      printf("initgraph error: %s.¥n",grapherrormsg(g_error));
      exit(1);
   }

   bar(0, 0, getmaxx()/2, getmaxy());

   getch();
   closegraph();
}
```

installuserdriver

機能　BGI デバイスドライバテーブルに，サードパーティ提供のデバイスドライバをインストールします。

形式

```
# include <graphics.h>
int far installuserdriver(char far * name, int huge (* detect)(void));
```

プロトタイプ　graphics.h

解説　**installuserdriver** を使用すると，サードパーティ供給のデバイスドライバを，BGI の内部テーブルに追加することができます。引数 *name* は新しいデバイスドライバファイル (.BGI) の名前です。引数 *detect* はオプションで，新しいドライバについてくる場合がある自動検出関数を指すポインタです。この自動検出関数は引数を持たず，関数値として整数を返します。
このサードパーティ供給ドライバの使う方法は2つあります。たとえば，いま Spiffy Graphics Array (SGA) という新しいグラフィックボードを購入し，SGA の製造元から BGI デバイスドライバ (SGA.BGI) も供給されたとしましょう。このドライバを使う最も簡単な方法は，まず **installuser-driver** を呼び出してこれをインストールし，その戻り値（割り当てられたドライバ番号）を直接 **initgraph** に渡すというものです。
もう1つは，より一般的で，**initgraph** によってハードウェア検出ロジックの一部として呼び出される自動検出関数（おそらく SGA の供給元はこの自動検出関数をつけてくるはずです）をリンクするという方法です。ドライバを（**installuserdriver** を呼び出して）インストールする際に，デバイスドライバファイルの名前とともにこの関数のアドレスを引数として渡します。

デバイスドライバファイル名とSGA自動検出関数をインストールした後は, **initgraph** を呼び出して通常の自動検出プロセスを行なわせます。**initgraph** は, 組み込みの自動検出関数(**detectgraph**)を呼び出す前に, SGA自動検出関数を呼び出します。SGA自動検出関数がSGAハードウェアを見つけられなかった場合は-11(grError)を返し, **initgraph** は引き続いて通常のハードウェア検出ロジックを呼び出します（他にもサードパーティ提供の自動検出関数が存在する場合は, それらもインストールされた順で呼ばれます)。自動検出関数は, SGAが存在すると認識した場合, 負でないモード番号を返し, **initgraph** はその値を用いてSGA.BGIを見つけてロードし, ハードウェアを自動検出関数が推奨するデフォルトのグラフィックスモードにして, 最後に制御をユーザプログラムに戻します。
一度に10個までのドライバをインストールすることができます。

戻り値

installuserdriver によって返される値は, **initgraph** が新しくインストールされたドライバを選択させるために渡すドライバ番号です。

可搬性

installuserdriver はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは, サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目

initgraph, **registerbgidriver**

例

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>

int Driver, Mode;

int huge detectSGA(void)     /* 自動検出ロジック */
{
   int found, defaultmode;

   /* 必要なハードウェアを検出する
      found = ・・・・・                    */

   if (!found)   /* 見つからなければエラーを返す */
      return (grError);

   /* デフォルトのグラフィックスモードを決定する
      defaultmode = ・・・・・                          */

   return (defaultmode);
}

main()
{
   Driver = installuserdriver("SGA", detectSGA);

   if ( grOk != graphresult() )  {     /* テーブルがいっぱい ? */
      printf("Error installing user driver SGA.¥n");
      exit(1);
   }

   Driver = DETECT;                    /* 自動検出を行なう */
   initgraph(&Driver, &Mode, "");  /* 検出を無効にする */

   if ( grOk != graphresult() )  exit(1);

   outtext("User Installed Drivers Supported");

   getchar();
   closegraph();
}
```

installuserfont

機能　BGIシステムには組み込まれていないフォントファイル(.CHR)をロードします。

形式
```
# include <graphics.h>
int far installuserfont(char far * name) ;
```

プロトタイプ　graphics.h

説明　*name*は，ストロークフォントを含むフォントファイルへのパス名です。一度に20個までのフォントをインストールすることができます。

戻り値　**installuserfont**は，**settextstyle**が対応するフォントを選べるように渡すフォントID番号を返します。内部フォントテーブルがいっぱいになっている場合は，-11（grError）を返します。

可搬性　**installuserfont**はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**settextstyle**

line

機能　2点を結ぶ直線を描きます。

形式　# include <graphics.h>

void far line(int *x1*, int *y1*, int *x2*, int *y2*);

プロトタイプ　graphics.h

解説　**line**は，カレントカラー，ラインスタイルおよびライン幅を用いて，カレント位置（CP）を変更せずに，指定された2点（*x1*,*y1*）と（*x2*,*y2*）を結ぶ直線を描きます。現在位置（CP）は更新されません。

戻り値　ありません。

可搬性　**line**はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**linerel**, **lineto**, **setcolor**, **setlinestyle**, **setwritemode**

linerel

機能　現在位置（CP）から相対距離で直線を描きます。

形式

```
# include <graphics.h>
void far linerel(int dx, int dy);
```

プロトタイプ　graphics.h

解説　**linerel** は，CP から，CP との相対距離（*dx,dy*）である点への直線を描きます。CP は（*dx,dy*）だけ移動します。

戻り値　ありません。

可搬性　**linerel** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**line**，**lineto**，**setcolor**，**setlinestyle**，**setwritemode**

lineto

機能　CP から（x,y）への直線を描きます。

形式　#include <graphics.h>
void far lineto(int *x*, int *y*)；

プロトタイプ　graphics.h

解説　lineto は，CP から（*x*,*y*）への直線を描き，CP を（*x*,*y*）に動かします。

戻り値　ありません。

可搬性　**lineto** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**line, linerel, setcolor, setlinestyle, setvisualpage, setwritemode**

moverel

機能　現在位置（CP）を相対距離で動かします。

形式

```
#include <graphics.h>
void far moverel(int dx, int dy);
```

解説　**moverel** は，現在位置（CP）を *x* 方向に ***dx***，*y* 方向に ***dy*** 移動します。

戻り値　ありません。

可搬性　**moverel** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**moveto**

moveto

機能 現在位置（CP）を（x,y）に移動します。

形式
```
#include <graphics.h>
void far moveto(int x, int y);
```

プロトタイプ graphics.h

解説 **moveto** は，現在位置（CP）を与えられた位置（x,y）に移動します。

戻り値 ありません。

可搬性 **moveto** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **moverel**

outttext

機能　ビューポート内に文字列を表示します。

形式　# include <graphics.h>
void far outtext(char far * *textstring*) ;

プロトタイプ　graphics.h

解説　**outtext** は，ビューポート内で現在設定されている位置合わせ，フォント，方向，大きさを使用して文字列を表示します。
outtext は，与えられた文字列 *textstring* を CP に表示します。テキストの水平方向の桁揃えの設定が LEFT_TEXT で，またテキストの方向が HORIZ_DIR の場合，CP の *x* 座標は **textwidth** (*textstring*) だけ増加します。そうでない場合は CP は変化しません。
何種類もフォントを用いる場合，コードの互換性を保つためには，関数 **textwidth**，**textheight** を用いて文字列の大きさを決定してください。

注意：**outtext** を使ってデフォルトフォントで文字列を書く場合，カレントビューポートの外へはみだしてしまう部分は出力されません。

注意：**outtext** は，グラフィックスモードでのみ使用できます。テキストモードでは使用できません。

戻り値　ありません。

可搬性　**outtext** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**gettextsettings**，**outtextxy**，**settextjustify**，**settextstyle**，**textheight**，**textwidth**

outttextxy

機能　指定した位置に文字列を表示します。

形式　# include <graphics.h>
void far outtextxy(int *x*, int *y*, char far * *textstring*);

プロトタイプ　graphics.h

解説　**outtextxy** は，ビューポート内で現在設定されている位置合わせ，フォント，方向，大きさを使用して，文字列を与えられた位置 (*x*,*y*) に表示します。
何種類もフォントを用いる場合，コードの互換性を保つためには，関数 **textwidth**，**textheight** を用いて文字列の大きさを決定してください。

注意：**outtextxy** を使ってデフォルトフォントで文字列を書く場合，カレントビューポートの外へはみだしてしまう部分は出力されません。

注意：**outtextxy** は，グラフィックスモードでのみ使用できます。テキストモードでは使用できません。

戻り値　ありません。

可搬性　**outtextxy** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**gettextsettings**，**outtext**，**textheight**，**textwidth**

pieslice

機能　パイスライス（扇形）を描いて，その中をフィルします。

形式　# include <graphics.h>
void far pieslice(int *x*, int *y*, int *stangle*, int *endangle*, int *radius*)；

プロトタイプ　graphics.h

解説　**pieslice** は，中心が(*x*, *y*)，半径が *radius* の扇形を描いて，その中をフィルします。扇形の角度は *stangle* から *endangle* までです。扇形の輪郭をカレントドロウカラーで描き，カレントフィルパターンとフィルカラーで中をフィルします。

pieslice の角度は，反時計まわりで，0度が時計で3時の方向，90度が12時の方向です。

注意［**IBM PC**］：CGA のハイレゾリューションモード，あるいはモノクログラフィックスアダプタを使用している場合，このマニュアルにあるサンプルプログラムは期待通りに動かないことがあります。CGA のハイレゾリューションモードまたはモクロアダプタを使っている場合は，フィルカラーあるいはドロウカラーをセットする関数（**setcolor**, **setfillstyle**, **setlinestyle** など）には，シンボリック定数を使わずに，値1を渡すようにしてください。**arc** 関数の**例 2** で，CGA あるいはモノクロアダプタ上で **pieslice** をどのように使うかを示しています。

戻り値　ありません。

可搬性　**pieslice** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**fillellipse**, **graphresult**, **sector**, **setfillstyle**

例　　　**arc** を参照してください。

putimage

機能　ビットイメージを画面上に出力します。

形式

```
#include <graphics.h>
void far putimage(int left, int top, void far * bitmap, int op);
```

プロトタイプ　graphics.h

解説

putimage は，**getimage** によって以前にセーブされたビットイメージを画面上に復元します。イメージの左上隅は（*left*,*top*）に置かれます。*bitmap* は，ビットイメージが格納されているメモリ内の領域を指します。

引数 *op* は，画面上のピクセルのカラーの計算で行なわれる演算の種類を指定します。この演算は画面上に既にあるピクセルとそれに対応するメモリ内のピクセルに基づいて行われます。

graphics.h で定義されている *putimage_ops* テーブルにこれらの演算の名前が与えられています。

名前	値	機能説明
COPY_PUT	0	コピー
XOR_PUT	1	排他的論理和
OR_PUT	2	論理和
AND_PUT	3	論理積
NOT_PUT	4	ソースの逆コピー

COPY_PUT は，ソースビットマップイメージを画面上に単にコピーします。

XOR_PUT は，ソースイメージと画面上のイメージとの XOR をとります。

OR_PUT は，ソースイメージと画面上のイメージの OR をとります。

以下同様です。

戻り値　　ありません。

可搬性　　**putimage** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　　**getimage, imagesize, putpixel, setvisualpage**

例　　**getimage** を参照してください。

putpixel

機能 指定された点のピクセルをプロットします。

形式 # include <graphics.h>

void far putpixel(int *x*, int *y*, int *color*);

プロトタイプ graphics.h

解説 **putpixel** は，**color** で定義されるカラーで（*x*,*y*）に点を描きます。

戻り値 ありません。

可搬性 **putpixel** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **getpixel**，**putimage**

rectangle

機能　長方形を描きます。

形式　#include <graphics.h>

void far rectangle(int *left*, int *top*, int *right*, int *bottom*)；

プロトタイプ　graphics.h

解説　**rectangle**は，カレントラインスタイル，ライン幅，カラーを使用して長方形を描きます。

(*left*,*top*)は長方形の左上隅を，(*right*,*bottom*)は右下隅を指定します。

戻り値　ありません。

可搬性　**rectangle**はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**bar**，**bar3d**，**setcolor**，**setlinestyle**

例

```
int i;
for (i=0; i<10; i++)
   rectangle(20-2*1, 20-2*1,10*(i+2),10*(i+2));
```

registerbgidriver

機能　ユーザがロードした，あるいはリンクされているグラフィックスドライバコードをグラフィックスシステムに登録します。

形式

```
#include <graphics.h>
int registerbgidriver(void (* driver)(void));
```

プロトタイプ　graphics.h

解説　**registerbgidriver** によって，ユーザがドライバファイルをロードして，そのドライバを登録することができます。ドライバのメモリ位置が **registerbgidriver** に渡されていれば，**initgraph** はその登録されたドライバを使用します。ユーザが登録するドライバは，ディスクからヒープ上にロードするか，(BINOBJ.EXE を使って).OBJ ファイルに変換して.EXE ファイルにリンクすることができます。

registerbgidriver を呼び出すと，グラフィックスシステムにリンクされるべきドライバが存在することを知らせることになります。このルーチンは，そのドライバに対するリンクされるべきコードをチェックし，コードが正しい場合は内部テーブルに登録します。リンクされるドライバに関しては，ユーザーズガイド付録Cの「**BGIOBJ**」の節で詳しく説明されています。

registerbgidriver の呼び出しでリンクされるドライバの名前を使用すると，コンパイラ（およびリンカ）に，そのパブリック名でオブジェクトファイルをリンクするよう指示することになります。

戻り値　**registerbgidriver** は，指定されたドライバが正しくない場合は，負のグラフィックスエラーコードを返します。そうでない場合は，内部ドライバ番号を返します。

ユーザ供給のドライバを登録する場合は，**registerbgidriver** が返した値を，使用するドライバ番号として **initgraph** に渡さなければなりません。

可搬性　**registerbgidriver** は PC-9801 シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**graphresult**，**initgraph**，**installuserdriver**，**registerbgifont**

例

```
/* PC-9801 EGCドライバを登録 */
if (regsterbgidriver(PC98EGC_driver) <0) exit(1);
```

registerbgifont

機能 リンクされるストロークフォントコードを登録します。

形式
#include <graphics.h>
int registerbgifont(void (* *font*)(void));

プロトタイプ graphics.h

解説 **registerbgifont** の呼び出しは，*font* によって指されるフォントがリンク時に取り込まれていることをグラフィックスシステムに知らせます。
このルーチンは，指定されたフォントに対するリンクされたコードをチェックし，コードが正しい場合はその内部テーブルに登録します。リンクされるフォントに関しては，ユーザーズガイド付録Cの「**BGIOBJ**」の節で詳しく説明されています。
registerbgifont を呼び出すときにリンクされるフォントの名前を使用すると，コンパイラ（あるいはリンカ）に，そのパブリック名でオブジェクトファイルをリンクするよう指示することになります。
ユーザ供給のフォントを登録する場合には，**registerbgifont** が返した値を，**settextstyle** に使用するフォント番号として渡さなければなりません。

戻り値 **registerbgifont** は，指定されたフォントが正しくない場合は，負のグラフィックスエラーコードを返します。そうでない場合は，登録したフォントのフォント番号を返します。

可搬性 **registerbgifont** は PC-9801 シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **graphresult**，**initgraph**，**installuserdriver**，**registerbgidriver**，**settextstyle**

例

```
/* ゴシックフォントを登録 */
if (registerbgifont(gothic_font) != GOTHIC_FONT) exit(1);
```

restorecrtmode

機能　画面モードを，**initgraph** 実行前のモードに戻します。

形式

```
#include <graphics.h>
void far restorecrtmode(void);
```

プロトタイプ　graphics.h

解説　**restorecrtmode** は，**initgraph** によって検出された元の画面モードを回復します。

restorecrtmode は，**setgraphmode** と組み合わせて，テキストモードとグラフィックモードの間を行き来するために使用することができます。**textmode** を同じ目的で使用することはできません。**textmode** は，画面がテキストモードのときにのみ使用でき，他のテキストモードに切り換えるときに使うことができます。

注意：IBM PC では，テキスト画面とグラフィックス画面を同時に表示することができないためにこうした関数が用意されています。PC-9801 では，同時表示が可能なので，**restorecrtmode** および **setgraphmode** を使わずに，グラフィックス出力とテキスト出力を混在させて，その両方を画面に表示させることができます。**restorecrtmode** および **setgraphmode** は，どちらも画面全体をクリアして初期化するので，PC-9801 でこれらを使う場合には，テキストとグラフィックスを同時に表示することはできません。

戻り値　ありません。

可搬性　**restorecrtmode** は PC-9801 シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getgraphmode**，**initgraph**，**setgraphmode**

sector

機能　楕円の扇形を描いて，その中をフィルします。

形式

```
#include <graphics.h>
void far sector(int x, int y, int stangle, int endangle,
                int xradius, int yradius);
```

プロトタイプ　graphics.h

解説　**sector** は，中心を（*x*,*y*），水平および垂直方向の半径をそれぞれ *xradius*, *yradius* とし，角度が *stangle* から *endangle* までの楕円の扇形を描いて，その中をフィルします。扇形の輪郭はカレントカラーで描かれ，**setfillstyle** および **setfillpattern** でセットされているパターンとカラーでフィルします。

sector に与える角度は，反時計回りで，3時の位置が0°，12時の位置が90°となります。

楕円の扇形をフィルしているときにエラーが起きた場合，**graphresult** は -6（grNoScanMem）を返します。

戻り値　ありません。

可搬性　**sector** は PC-9801 シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**arc**，**circle**，**ellipse**，**getarccoords**，**getaspectratio**，**pieslice**，**setfillpattern**，**setfillstyle**，**setgraphbufsize**

setactivepage

機能 グラフィックス出力を行なうアクティブページをセットします。

形式 # include <graphics.h>
void far setactivepage(int *page)*；

プロトタイプ graphics.h

解説 **setactivepage** は，*page* をアクティブグラフィックスページにセットします。これ以降，グラフィックス出力はグラフィックスページ *page* に対して行なわれます。
使用しているシステムがグラフィックスページを何枚持っているかによって，アクティブグラフィックスページが画面に見えている場合と見えていない場合があります。

注意：この関数はデバイスを直接制御するもので，現在使用中のシステムにページが何ページ存在するかは得ることができません。PC-9801 では，初期型の PC-9801 と PC-9801U はグラフィックスページを 1 画面しか持っていません。他のタイプはグラフィックスページを 2 枚持っており，1 枚目が 0 ページ，2 枚目が 1 ページになります。

戻り値 ありません。

可搬性 **setactivepage** は PC-9801 シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **setvisualpage**

例

```
cleardevice();
setvisualpage(0);          /* 0ページをビジュアルに */
setactivepage(1);          /* 1ページを出力用に使う */
bar(50, 50, 150, 150);     /* 1ページにバーを描画   */
setvisualpage(1);          /* 1ページをビジュアルに */
```

setallpalette

機能 すべてのパレットカラーを指定どおりに変更します。

形式 # include <graphics.h>

void far setallpalette(struct palettetype far * *palette*) ;

プロトタイプ graphics.h

解説 **setallpalette** は，*palette* が指す **palettetype** 型構造体の内容にしたがって，カレントパレットを変更します。

EGA/VGA のパレット中のカラーは，**setallpalette** によって部分的に(あるいは全部を) 変更することができます。

setallpalette で用いられる定数 MAXCOLORS と **palettetype** 型構造体は，graphics.h の中で次のように定義されています。

```
#define MAXCOLORS  15

struct palettetype {
    unsigned char size;
    signed char colors[MAXCOLORS + 1];
};
```

size は，カレントグラフィックスドライバと，カレントグラフィックスモードでの，パレット中のカラー数です。

colors は，パレット中の各項目に対応する実際のカラー番号を要素とする *size* バイトの配列です。*colors* の要素が-1 の場合は，その項目のパレットカラーは変更されません。**setallpalette** で使われる配列 *colors* の要素は，graphics.h の中で定義されている以下のシンボリック定数で表わすことができます。

PC-9801 / CGA		EGA / VGA	
名前	**値**	**名前**	**値**
BLACK	0	EGA_BLACK	0

BLUE	1	EGA_BLUE	1
GREEN	2	EGA_GREEN	2
CYAN	3	EGA_CYAN	3
RED	4	EGA_RED	4
MAGENTA	5	EGA_MAGENTA	5
BROWN	6	EGA_LIGHTGRAY	7
LIGHTGRAY	7	EGA_BROWN	20
DARKGRAY	8	EGA_DARKGRAY	56
LIGHTBLUE	9	EGA_LIGHTBLUE	57
LIGHTGREEN	10	EGA_LIGHTGREEN	58
LIGHTCYAN	11	EGA_LIGHTCYAN	59
LIGHTRED	12	EGA_LIGHTRED	60
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61
YELLOW	14	EGA_YELLOW	62
WHITE	15	EGA_WHITE	63

指定できるカラーはカレントグラフィックスドライバとグラフィックスモードに依存していることに注意してください。
パレットを変更すると，変化は即座に画面に反映されます。パレットのカラーが変化するたびに，画面上でそのカラーが表示されている部分はすべて新しいカラーに変わります。

注意：**setallpalette**は，IBM8514ドライバでは使用できません。IBM8514では**setrgbpalette**を使ってください。

戻り値　ありません。
setallpaletteに正しくない入力を与えると，**graphresult**は-11を返し，カレントパレットは変更されません。

可搬性　**setallpalette**はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getpalette**，**graphresult**，**setbkcolor**，**setcolor**，**setpalette**

setaspectratio

機能　デフォルトのアスペクト比補正因子を変更します。

形式　# include <graphics.h>

void far setaspectratio(int *xasp*, int *yasp*);

プロトタイプ　graphics.h

解説　**setaspectratio** は，グラフィックスシステムのデフォルトのアスペクト因子を変更するために使用します。アスペクト比は，グラフィックスシステムが円を真円として描くことができるようにするために使用されます。画面に出力した円が楕円になってしまうとすれば，モニタのピクセルの配置が適切でないと考えることができます。このような場合，モニタを配置しなおしてハード的に補正することができますが，ソフト的に **setaspectratio** を使ってアスペクト比を変更して補正することも可能です。現在のアスペクト比は，**getaspectratio** の呼び出しによって得ることができます。

戻り値　ありません。

可搬性　**setaspectratio** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**circle**，**getaspectratio**

setbkcolor

機能　パレットを用いてカレントバックグラウンドカラーをセットします。

形式

```
#include <graphics.h>
void far setbkcolor(int color);
```

プロトタイプ　graphics.h

解説　**setbkcolor** は，バックグラウンドを *color* で指定されたカラーにセットします。引数 *color* には，graphics.h で定義されている，次に示す名前あるいは番号を使うことができます。

番号	名前	番号	名前
0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

注意：PC-9801の8色モード(PC98C8)でも，上の表のすべての名前あるいは番号を指定することはできますが，実際に表示される色は8色になります。8色モードの場合，上の表で同じ行に示されているカラーは同じ意味を持つことになります。

たとえば，バックグラウンドカラーを青にしたいときには，**setbkcolor** を次のように呼び出します。

```
setbkcolor(BLUE)  /* または */  setbkcolor(1)
```

CGA および EGA システムでは，**setbkcolor** は，パレット中の最初の項目を変更することによってバックグラウンドカラーを変更します。

注意：EGA または VGA システムで **setpalette** あるいは **setallpalette** によってパレットカラーを変更する場合，定義済みのシンボリック定数は正しいカラーを示していないことがあります。これは，**setbkcolor** へのパラメータが，特定のカラーではなく，カレントパレット中の項目番号を示しているためです(渡されたパラメータが0でなければ，バックグラウンドカラーは常に黒にセットされます)。

戻り値　ありません。

可搬性　**setbkcolor** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getbkcolor**，**setallpalette**，**setcolor**，**setpalette**

setcolor

機能　パレットを用いてカレントドロウカラーをセットします。

形式

```
# include <graphics.h>
void far setcolor(int color);
```

プロトタイプ　graphics.h

解説

setcolor は，カレントドロウカラーを *color* に設定します。*color* の範囲は0から **getmaxcolor** までです。

ドロウカラーとは，線などを描くときにピクセルにセットされる値のことです。PC-9801シリーズおよびIBM PC の EGA では，次の表に示すドロウカラーを使うことができます。

番号	名前	番号	名前
0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

ドロウカラーを選択するには，そのカラー番号あるいは対応するシンボリック名を setcolor に渡して呼び出します。たとえば PC-9801でドロウカラーに紫を選択したいとれば，**setcolor**(MAGENTA)，あるいは **setcolor**(5) とします。

注意：PC-9801の8色モード(PC98C8)でも，上の表のすべての名前あるいは番号を指定することはできますが，実際に表示される色は8色になります。8色モードの場合，上の表で同じ行に示されているカラーは同じ意味を持つことになります。

CGAシステムでは以下のカラーが指定可能です。

パレット番号	カラー番号（ピクセル値）に割り当てられている定数 1	2	3
0	CGA_LIGHTGREEN	CGA_LIGHTRED	CGA_YELLOW
1	CGA_LIGHTCYAN	CGA_LIGHTMAGENTA	CGA_WHITE
2	CGA_GREEN	CGA_RED	CGA_BROWN
3	CGA_CYAN	CGA_MAGENTA	CGA_LIGHTGRAY

たとえばCGA0モードでは，パレットには，バックグラウンドカラー，明るい緑，明るい赤，黄色の4色があります。このモードで黄色を選択したいとすれば，**setcolor**(3)，あるいは**setcolor**(CGA_YELLOW)を呼び出します。

戻り値　ありません。

可搬性　**setcolor**はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PCでは，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**getcolor**，**getmaxcolor**，**setallpalette**，**setbkcolor**，**setpalette**

setfillpattern

機能　ユーザ定義のフィルパターンを定義します。

形式　# include <graphics.h>

void far setfillpattern(char far * *upattern*, int *color*) ;

プロトタイプ　graphics.h

解説　**setfillpattern** は, **setfillstyle** と似ていますが, こちらはシステム定義のパターンではなく，ユーザ定義の8×8パターンを設定するために使用します。*upattern* は8バイト列を指すポインタです。各バイトはパターン中の8ピクセルに対応しています。パターン内で1に設定されているビットに対応するピクセルがプロットされます。

戻り値　ありません。

可搬性　**setfillpatern** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**getfillpattern**, **getfillsettings**, **setfillstyle**

setfillstyle

機能 フィルパターンとフィルカラーをセットします。

形式

```
#include <graphics.h>
void far setfillstyle(int pattern, int color);
```

プロトタイプ graphics.h

解説 **setfillstyle** は，カレントフィルパターンとフィルカラーを設定します。ユーザ定義のフィルパターンを設定するには，**setfillstyle** に12（USER FILL）の *pattern* を与えるのではなく，**setfillpattern** を呼び出してください。

graphics.h で定義されている *fill_pattern* テーブルは，システム定義フィルパターンの名前とユーザ定義パターンの指示子を与えています。

名前	値	機能説明
EMPTY_FILL	0	バックグラウンドカラーでフィル(中空)
SOLID_FILL	1	ベタ塗り
LINE_FILL	2	横線でフィル
LTSLASH_FILL	3	///でフィル
SLASH_FILL	4	///でフィル，太線
BKSLASH_FILL	5	＼＼＼でフィル，太線
LTBKSLASH_FILL	6	＼＼＼でフィル
HATCH_FILL	7	淡いハッチ（格子）でフィル
XHATCH_FILL	8	濃いクロスハッチ(斜めの格子)でフィル
INTERLEAVE_FILL	9	インターリーブ線でフィル
WIDE_DOT_FILL	10	すき間が広い点でフィル
CLOSE_DOT_FILL	11	すき間が狭い点でフィル
USER_FILL	12	ユーザ定義のフィルパターン

EMPTY_FILL 以外のものは，カレントフィルカラーでフィルします。EMPTY_FILL は，カレントバックグラウンドカラーを用います。**setfillstyle** に不適当な値が渡されると，**graphresult** は-11（grError）を返し，フィルパターンとフィルカラーは変更されず元のままになります。

戻り値　ありません。

可搬性　**setfillstyle** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連事項　**bar**，**bar3d**，**fillpoly**，**floodfill**，**getfillsettings**，**graphresult**，**pieslice**，**sector**，**setfillpattern**

setgraphbufsize

機能　内部グラフィックスバッファのサイズを変更します。

形式　# include <graphics.h>

unsigned far setgraphbufsize(unsigned *bufsize*);

プロトタイプ　graphics.h

解説　グラフィックスルーチンの中には（**floodfill** のように）メモリバッファを用いるものがあります。このメモリバッファは **initgraph** が呼ばれたときに確保され，**closegraph** が呼ばれたときに解放されます。**_graphgetmem** によって確保されるバッファサイズのデフォルトは4096バイトです。

このバッファサイズを小さくしたいとき(メモリを節約する)，あるいは大きくしたいとき（たとえば **floodfill** を呼び出してエラー値-7，すなわちフラッドフィルでメモリが不足した場合）にはこの関数を使うことになります。**setgraphbufsize** は，**initgraph** に，**_graphgetmem** を呼ぶ際に内部バッファとしてどれだけのメモリを確保すべきかをつたえます。

注意：**setgraphbufsize** は，**initgraph** より前に呼び出さなければなりません。**initgraph** が呼び出された後で行なわれたすべての **setgraphbufsize** の呼び出しは，次に **closegraph** が呼び出されるまでは意味をなしません。

戻り値　**setgraphbufsize** は内部バッファの以前のサイズを返します。

可搬性　**setgraphbufsize** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**closegraph**，**_graphfreemem**，**_graphgetmem**，**initgraph**

例

```
int cbsize;
cbsize = setgraphbufsize(1000);      /* カレントサイズを得る */
setgraphbufsize(cbsize);             /* 元のサイズに戻す     */
printf("The graphics buffer is currently %d bytes.", cbsize);
```

setgraphmode

機能 システムをグラフィックモードにセットし，テキスト画面をクリアします。

形式

```
# include <graphics.h>
void far setgraphmode(int mode) ;
```

プロトタイプ graphics.h

解説 **setgraphmode** は，**initgraph** で設定されるデフォルトのモードとは違ったグラフィックスモードを選択します。*mode* はカレントデバイスドライバに有効なモードでなければなりません。**setgraphmode** は画面をクリアし，すべてのグラフィックス設定をデフォルトにセットします(CP，パレット，カラー，ビューポートなど)。**setgraphmode** は **restorecrtmode** とともに，テキストモードとグラフィックスモードの切り替えに使用します。

注意：IBM PC では，テキスト画面とグラフィックス画面を同時に表示することができないためにこうした関数が用意されています。PC-9801では，同時表示が可能なので，**restorecrtmode** および **setgraphmode** を使わずに，グラフィックス出力とテキスト出力を混在させて，その両方を画面に表示させることができます。**restorecrtmode** および **setgraphmode** は，どちらも画面全体をクリアして初期化するので，PC-9801でこれらを使う場合には，テキストとグラフィックスを同時に表示することはできません。

戻り値 カレントデバイスドライバに不適当なモードを **setgraphmode** に与えた場合，**graphresult** は-10(grInvalidMode)の値を返します。

可搬性 **setgraphmode** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目 **getgraphmode**，**getmoderange**，**graphresult**，**initgraph**，**restorecrtmode**

setlinestyle

機能　カレントライン幅とラインスタイルをセットします。

形式　# include <graphics.h>

void far setlinestyle(int *linestyle*, unsigned *upattern*, int *thickness*);

プロトタイプ　graphics.h

解説　**setlinestyle** は，**line**，**lineto**，**rectangle**，**drawpoly** などで描かれるすべてのラインスタイルを設定します。

linesettingstype 構造体は，graphics.h で次のように定義されています。

```
struct linesettingstype {
    int linestyle;
    unsigned upattern;
    int thickness;
};
```

linestyle は，それ以降に描かれるラインのスタイルを指定します（たとえば，実線，点線，一点鎖線，破線）。graphics.h で定義されている *line_style* テーブルには，これらの演算子の名前が与えられています。

名前	値	機能
SOLID_LINE	0	実線
DOTTED_LINE	1	点線
CENTER_LINE	2	一点鎖線
DASHED_LINE	3	破線
USERBIT_LINE	4	ユーザ定義のラインスタイル

thickness は，それ以降に描かれる線の幅が，普通か太いかを指定するものです。

名前	値	機能
NORM_WIDTH	1	1ピクセル幅
THICK_WIDTH	3	3ピクセル幅

upattern は，*linestyle* が USERBIT_LINE(4)のときにのみ適用する16ビットパターンです。この場合，パターンワード内の1のビットに対応するライン内のピクセルが，カレントドロウカラーで描かれます。たとえば，実線は0xFFFF の *upattern*(すべてのピクセルの描画)に対応し，破線は0x3333または0x0F0F の *upattern* に対応します。**setlinestyle** の引数 *linestyle* が USERBIT_LINE(4)でない場合も，*upattern* は与えなければなりません（無視されますが)。

注意：*linestyle* パラメータは，円弧，円周，楕円弧，楕円周には影響を与えません(扇形と楕円の扇形の半径には影響を与えます)。*thickness* パラメータはこれらに影響します。

戻り値

setlinestyle に不適当な値が渡された場合，**graphresult** は-11を返し，カレントラインスタイルは変化しません。

可搬性

setlinestyle は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目

bar3d, **getlinesettings**, **graphresult**, **line**, **linerel**, **lineto**, **rectangle**

setnewdriver（PC-9801のみ）

機能　デフォルト以外のグラフィックスドライバを登録します。

形式
```
# include <graphics.h>
int far setnewdriver("PC98", detect98) ;
```

プロトタイプ　graphics.h

解説　**setnewdriver** は，バージョン1.5のコードとの互換性のためにのみ用意されているもので，内部では単にそのままの引数で **installuserdriver** を呼び出すだけです。

バージョン2.0では，PC-9801のドライバもあらかじめ BGI の内部テーブルに登録されているので，**setnewdriver** を呼び出す必要はなくなっています。

詳しくは，**initgraph** および **installuserdriver** を参照してください。

戻り値　**installuserdriver** と同じです。

可搬性　PC-9801シリーズでのみ動作します。

関連項目　**installuserdriver**

setpalette

機能 パレットカラーの1つを変更します。

形式 # include <graphics.h>
void far setpalette(int *colornum*, int *color*) ;

プロトタイプ graphics.h

解説 **setpalette** は，パレットの中の *colornum* 番目の項目を，*color* に変更します。たとえば **setpalette**(0,5)は，カレントパレットの最初のカラー(バックグラウンドカラー) を，実際のカラー番号5のカラーに変更します。*size* がカレントパレットに含まれる項目の数とすると，*colornum* は0から(*size*-1)との間の値をとることができます。

PC-9801および IBM PC の EGA/VGA のパレット中のカラーは，**setpalette** によって，1項目ごとにセットすることができます。CGA では，**setpalette** で変更できるのはパレット中の最初の項目(*colornum*=0, バックグラウンドカラー) だけです。

setpalette に渡す *color* パラメータは，graphics.h の中で定義されている以下のシンボリック定数で表わすことができます。

PC-9801 / CGA		EGA / VGA	
名前	**値**	**名前**	**値**
BLACK	0	EGA_BLACK	0
BLUE	1	EGA_BLUE	1
GREEN	2	EGA_GREEN	2
CYAN	3	EGA_CYAN	3
RED	4	EGA_RED	4
MAGENTA	5	EGA_MAGENTA	5
BROWN	6	EGA_LIGHTGRAY	7
LIGHTGRAY	7	EGA_BROWN	20

DARKGRAY	8	EGA_DARKGRAY	56
LIGHTBLUE	9	EGA_LIGHTBLUE	57
LIGHTGREEN	10	EGA_LIGHTGREEN	58
LIGHTCYAN	11	EGA_LIGHTCYAN	59
LIGHTRED	12	EGA_LIGHTRED	60
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61
YELLOW	14	EGA_YELLOW	62
WHITE	15	EGA_WHITE	63

指定できるカラーはカレントグラフィックスドライバとグラフィックスモードに依存していることに注意してください。
パレットを変更すると，変化は即座に画面に反映されます。パレットカラーが変化するたびに，画面上のそのカラーが表示されている部分はすべて新しいカラーに変わります。

注意：IBM8514ドライバでは **setpalette** は使用できません。IBM8514では，**setrgbpalette** を使用してください。

正しくない入力を **setpalette** に与えると，**graphresult** は-11（grError）を返し，カレントパレットは変更されません。

戻り値　ありません。

可搬性　**setpalette** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**getpalette**，**graphresult**，**setallpalette**，**setbkcolor**，**setcolor**，**setrgbpalette**

setrgbpalette (PC-9801)

機能　ハードウェアパレットの実際の色を定義します。

形式　# include <graphics.h>

```
void far setrgbpalette(int colornum, int red, int green, int blue);
```

プロトタイプ　graphics.h

解説　**setrgbpalette** は，*colornum* で指定されたパレットの実際の色を *red*, *green*, *blue* で指定した値に変更します。
この関数により PC-9801シリーズの16色モードでは，4096色中から任意の16色を選んで使うことができます。
red, *green*, *blue* の値の有効範囲はカレントグラフィックスモードに依存しています。PC-9801ではこれらの値は以下のようになります。

	8色モード (PC98C8)	16色モード (PC98C16)
red	0～1	0～15
green	0～1	0～15
blue	0～1	0～15

戻り値　ありません。

可搬性　**setrgbpalette** は PC-9801シリーズでのみ動作します。

関連項目　**setpalette**

setrgbpalette (IBM PC)

機能　IBM8514のカラーを定義します。

形式　# include <graphics.h>
void far setrgbpalette(int *colornum*, int *red*, int *green*, int *blue*);

プロトタイプ　graphics.h

解説　**setrgbpalette** は，IBM8514および VGA ドライバにおいて使用することができます。
colornum はロードするパレット項目を定義し，*red*，*green*，および *blue* はパレット項目の色要素を定義します。
IBM8514ディスプレイ（および256K カラーモードでの VGA）では，*colornum* の範囲は0～255になります。VGA の他のモードでは，*colornum* の範囲は0～15です。*red*，*green*，および *blue* は下位バイトのみが使用され，その下位バイトの上位6ビットだけがパレットにロードされます。

注意：他の IBM グラフィックスアダプタとの互換性のために，BGI ドライバは，IBM8514の先頭から16個のパレット項目を EGA/VGA のデフォルトカラーと同じものに定義しています。これらの値はそのまま使用することができ，また，**setrgbpalette** で変更することもできます。

戻り値　ありません。

可搬性　**setrgbpalette** は，サポートするグラフィックスディスプレイアダプタを備えた IBM PC およびその互換機でのみ動作します。

関連項目　**setpalette**

settextjustify

機能 テキスト出力の位置合わせを設定します。

形式 # include <graphics.h>

void far settextjustify(int *horiz*, int *vert*);

プロトタイプ graphics.h

解説 **settextjustify** の呼び出し後に出力されるテキストは，CP に対して縦方向および横方向に指定された位置合わせ（テキストのどこを CP に合せて出力するか）が行なわれます。デフォルトの位置合わせの設定は LEFT_TEXT（水平方向），TOP_TEXT（垂直方向）です。

graphics.h の中の列挙型 *text_just* は，**settextjustify** に渡される horiz と vert の設定に使う名前を定義しています。

名前	値	方向
LEFT_TEXT	0	水平
CENTER_TEXT	1	水平および垂直
RIGHT_TEXT	2	水平
BOTTOM_TEXT	0	垂直
TOP_TEXT	2	垂直

horiz が LEFT_TEXT に等しく，*direction* が HORIZ_DIR の場合には，CP の *x* 座標は，**outtext**(*string*) を呼び出した後では，**textwidth**(*string*) だけ増加することになります。これ以外の位置合わせの設定の場合には CP は変化しません。

settextjustify は，**outtext** および **outtextxy** によって書かれるテキストに効果を持ちます。テキストモードやストリーム関数に対しては意味を持ちません。

settextjustify に無効な入力を渡すと，**graphresult** は-11 (grError) を返し，カレントテキストの位置合わせの指定は変化しません。

戻り値　ありません。

可搬性　**settextjustify** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　**gettextsettings**，**graphresult**，**outtext**，**settextstyle**

settextstyle

機能　グラフィックス出力のカレントテキスト属性をセットします。

形式
```
#include <graphics.h>
void far settextstyle(int font, int direction, int charsize);
```

プロトタイプ　graphics.h

解説

settextstyle は，テキストフォント，テキストの表示される向き，文字のサイズをセットします。**settextstyle** の呼び出しは，**outtext** および **outtextxy** によって出力されるテキストすべてに影響を与えます。

settextstyle に渡されるパラメータ *font*, *direction*, *charsize* は次に示すような意味を持ちます。

font：1組の8×16ビットマップフォント（IBM PC では8×8）と，いくつかのストロークフォントが使用できます。8×16（または8×8）ビットマップフォントがデフォルトです。graphics.h の中で定義されている列挙型 *font_names* により，こうした種々のフォントの設定に用いる名前が与えられています（次に示す表を参照してください）。

名前	値	説明
DEFAULT_FONT	0	8×16（8×8）ビットマップフォント
TRIPLEX_FONT	1	ストローク・トリプレックスフォント
SMALL_FONT	2	ストローク・スモールフォント
SANSSERIF_FONT	3	ストローク・サンスセリフフォント
GOTHIC_FONT	4	ストローク・ゴシックフォント

デフォルトのビットマップフォントはそのまま使用できます。ストロークフォントは*.CHRというディスクファイルに蓄えられており，一度にひとつしかメモリに置けません。そのため，ストロークフォント（一番最後に選んだストロークフォントと別の種類の物）を選ぶときには，対応する*.CHRファイルをディスクから読み込まなくてはいけません。いくつかのストロークフォントを使用するとき，この読み込みを避けるためには，プログラムにフォントファイルをリンクすることになります。BGIOBJユーティリティでこれをオブジェクトファイルに変換し，**registerbgifont**でそれを登録するという手順でリンクすることができます（BGIOBJについてはユーザーズガイド付録Cで説明しています）。
漢字が使用できるのはデフォルトのビットマップフォントだけです。

direction：サポートされているフォントの向きは，水平テキスト（左から右）と垂直テキスト（反時計回りに90度回転した向き）です。デフォルトの方向はHORIZ_DIRです。

名前	値	方向
HORIZ_DIR	0	左から右へ
VERT_DIR	1	下から上へ

charsize：文字の大きさは *charsize* 因子を使って拡大することができます。*charsize* が0でない場合は，ビットマップおよびストローク文字が影響を受けます。*charsize* が0の場合は，ストローク文字だけが影響を受けます。

- *charsize* が1の場合，**outtext** と **outtextxy** は画面上の8×16ピクセルの長方形の領域に，8×16ビットマップフォントの文字を表示します。
- *charsize* が2の場合，これらの出力関数は16×32ピクセルの長方形の領域に，8×16ビットマップフォントの文字を表示します(以下同様に標準サイズの10倍 *charsize*＝10 まで可能です)。
- *charsize* が0の場合，出力関数 **outtext** と **outtextxy** は，デフォルトの文字拡大因子4 (4倍)，あるいは **setusercharsize** によって与えられるユーザ定義の文字サイズのどちらかを使用して，ストロークフォントテキストを拡大します。

テキストの実際の大きさを調べるには，必らず **textheight** と **textwidth** を使用してください。

戻り値　　ありません。

可搬性　　**settextstyle** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません)。

関連項目　　**gettextsettings**，**graphresult**，**installuserfont**，**settextjustify**，**setusercharsize**，**textheight**，**textwidth**

setusercharsize

機能 ストロークフォント文字の幅と高さを変更します。

形式 # include <graphics.h>
void far setusercharsize(int *multx*, int *divx*, int *mutly*, int *divy*);

プロトタイプ graphics.h

解説 **setusercharsize** は，グラフィックス関数で使用されるストロークフォントによるテキストのサイズを制御します。**setusercharsize** によってセットされる値は，**settextstyle** の *charsize* パラメータが0のときにのみ有効です。したがって，**setusercharsize** の前に **settextstyle** を呼び出して *charsize* を0にセットしておく必要があります。
setusercharsize を使って，幅と高さを決定するスケール因子を指定します。デフォルト幅は *multx/divx* 倍され，デフォルトの高さは *multy/divy* 倍されます。たとえば，テキストの幅を2倍に，高さを50%高くするには次のようにします。

```
multx = 2;  divx = 1;
multy = 3;  divy = 2;
```

戻り値 ありません。

可搬性 **setusercharsize** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **gettextsettings**, **graphresult**, **settextstyle**

例

```
#include <graphics.h>
#inckude <conio.h>

main()
{
   int graphdriver = DETECT, graphmode; /* 自動検出を要求 */
   char *title = "TEXT in a BOX";

   initgraph(&graphdriver, &graphmode, "");     /* 初期化 */

   settextjustify(CENTER_TEXT, CENTER_TEXT);
   setusercharsize(1,1,1,1);
   settextstyle(TRIPLEX_FONT, HORIZ_DIR, USER_CHAR_SIZE);
   setusercharsize(200, textwidth(title), 100, textheight(title));
   rectangle(0, 0, 200, 100);
   outtextxy(100, 50, title);
   getche();
   closegraph();
}
```

setviewport

機能　グラフィックス出力用のカレントビューポートを設定します。

形式　# include <graphics.h>

void setviewport(int *left*, int *top*, int *right*, int *bottom*, int *clip*)；

プロトタイプ　graphics.h

解説　**setviewport** は，グラフィックス出力用の新しいビューポートを作ります。ビューポートの四隅は，絶対画面座標の(*left*,*top*)と(*right*,*bottom*)によって与えられます。現在位置（CP）はビューポートの（0,0）へ移されます。

引数 *clip* は，描画がカレントビューポートの境界でクリッピングされる(端が切り取られる)かどうかを決めます。プログラムの中で **setviewport** を呼び出したときに *clip* が0でない場合は，すべての描画はカレントビューポートによってクリッピングされます。

無効な入力が **setviewport** に渡されると，**graphresult** は-11を返し，カレントビューポートの設定は変化しません。

戻り値　ありません。

可搬性　**setviewport** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**clearviewport**, **getviewsettings**, **graphresult**

setvisualpage

機能 ビジュアルグラフィックスページ番号をセットします。

形式
```
#include <graphics.h>
void far setvisualpage(int page);
```

プロトタイプ graphics.h

解説 **setvisualpage** は，*page* をビジュアルグラフィックスページにセットします。

戻り値 ありません。

可搬性 **setvisualpage** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **graphresult**，**setactivepage**

例 **setactivepage** を参照してください。

setwritemode

機能　グラフィックスモードにおける描線の書き込みモードを設定します。

構文　# include <graphics.h>
void far setwritemode(int *mode)*；

プロトタイプ　graphics.h

説明　次の2つの定数が定義されています。

```
COPY_PUT    = 0     /* MOV */
XOR_PUT     = 1     /* XOR */
```

各定数は，線の各ピクセルと画面の対応するピクセルとの間の2進演算に対応しています。すなわち，COPY_PUT はアセンブリ言語の MOV 命令を使って，画面上にあるものすべてに対して線を上書きします。XOR_PUT は，XOR 命令を使って，画面上にあるものと線に対して排他的 OR 操作を行ないます。連続して2回 XOR 命令を行うと，線は消えることになり，画面は元通りになります。

注意：**setwritemode** は現在のところ，**line**，**linerel**，**lineto**，**rectangle**，**drawpoly** に対してだけ有効です。

戻り値　ありません。

可搬性　**setwritemode** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**drawpoly**，**line**，**linerel**，**lineto**，**putimage**

textheight

機能　文字列の高さをピクセル単位で返します。

形式

```
# include <graphics.h>
int textheight(char far * textstring);
```

プロトタイプ　graphics.h

解説　グラフィックス関数 **textheight** は，カレントフォントサイズと拡大因子を用いて文字列 *textstring* の高さをピクセル単位で求めます。
この関数は，行間のスペースを調整したり，ビューポートの高さを計算したり，グラフ上やボックスの中にフィットするタイトルの大きさを求めたい場合に便利です。
たとえばPC-9801では，8×16のビットマップフォントで，拡大因子が1（**settextstyle** でセットされる）のとき，文字列"Turbo C"は16ピクセルの高さになります。IBM PC では，8×8ビットマップフォントで拡大因子が1であれば，"Turbo C"の高さは8ピクセルになります。
textheight を使うと，いちいち手計算をしなくても，文字列の高さを求めることができます。この関数を使えば，違ったフォントが選ばれた場合でも，ソースコードを変更する必要がないようにすることもできます。

戻り値　**textheight** は，テキストの高さをピクセル単位で返します。

可搬性　**textheight** はPC-9801シリーズ（またはIBM PCとその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目　**gettextsettings**，**outtext**，**outtextxy**，**settextstyle**，**textwidth**

textwidth

機能 文字列の幅をピクセル単位で返します。

形式 # include <graphics.h>

int far textwidth(char far * *textstring*);

プロトタイプ graphics.h

解説 **textwidth** は，文字列の長さ，カレントフォントサイズ，拡大因子を用いて，文字列 *textstring* の幅を求めます。

この関数は，ビューポートの幅を計算したり，グラフ上やボックスの中にフィットするタイトルの大きさを求めたい場合に便利です。

textwidth を使うと，いちいち手計算をしなくても，文字列の幅を求めることができます。この関数を使えば，違ったフォントが選ばれた場合でも，ソースコードを変更する必要がないようにすることもできます。

戻り値 **textwidth** は，テキストの高さをピクセル単位で返します。

可搬性 **textwidtht** は PC-9801シリーズ（または IBM PC とその互換機）でのみ動作します（IBM PC では，サポートするグラフィックスディスプレイアダプタを備えていなければなりません）。

関連項目 **gettextsettings**，**outtext**，**outtextxy**，**settextstyle**，**textheight**

PC-9801 の ROM-BIOS インターフェース

PC-9801用 ROM-BIOS インターフェースルーチンは，マシンの ROM チップに組み込まれているハードウェア基本制御ルーチンを駆動するためのインターフェースルーチンです。これらの関数を使用すれば，マシンの機能を最大限に引き出すことができます。ただし，他機種との互換性はまったくないので，可搬性のあるプログラムを作成しようとするのであれば，このインターフェースは使用すべきではありません。ここで説明するすべての関数は，PC-9801シリーズでのみ使用することができます。

ROM-BIOS インターフェースを使うと，次のようなことが行なえます。

■システム情報を得る（**bios98equip** など）
■キーボードインターフェース（**bios98key**）
■ディスク装置の操作（**bios98disk** など）
■プリンタの制御（**bios98print**）
■ RS-232C の制御（**bios98com**，**bios98com_init** など）
■マウスの制御（**bios98mouse** など）
■インターバルタイマの制御（**bios98timer** など）
■日付時刻の読み出しと設定（**bios98time**）
■ ROM のフォントパターンの読み出しとユーザ定義文字の設定（**getfont** など）

なお，インターフェースの詳細な仕様については，各機種の「**ユーザーズマニュアル**」およびアスキー出版局の「**PC-9800 シリーズ テクニカルデータブック**」などを参照してください。

パケット構造体

各インターフェースルーチンでは，BIOS とのデータのやりとりを，構造体を通して行ないます(構造体を使用しない関数もありますが)。この節では，こうした構造体のことを**パケット**と総称しています。各ルーチンを使用する際には，パケット（構造体）の所定の位置（メンバ）に必要なパラメータを入れ，そのパケットを引数として関数を呼び出します。

パケット構造体の最初の2つのメンバは，すべての BIOS 関数において共通で，1番目が機能コード（メンバ名 cmmd），2番目がステータス（stus）となっています。cmmd には

ROM ルーチンに与える機能コードを指定します。stus は，ROM ルーチンを呼び出した後の状態（ステータス）で，BIOS の返した値が格納されます（stus を使用しない関数もあります）。

パケット構造体の3番目以降のメンバは，各関数によって異なり，対応する ROM ルーチンに応じたパラメータを指定したり，ROM ルーチンが返した値を格納したりします。

パケットの解説では，次のような形式の表を使って説明しています（ただし，**bios98disk** および **bios98harddisk** は，これとはやや異なります。これらについては該当のページを見てください）。

PACKET_NAME		1	2	3	4	.
cmmd	AH	R	R	R	R	..
stus	AH	W	W	W	W	..
info_1	CX	W	×	×	×	..
info_2	ALCH	×	R	W	×	..
info_3	CL	×	Rx	Rx	×	..
info_4	ALCH	×	RW	RW	R	..
...	..	..	..	..	..	..

表の一番上の行は，構造体の型名と使用できる機能コード（cmmd に指定する値）です。機能コードは10進数で表わしています。

2行目以下は左から，メンバ名，（その関数あるいは BIOS が使用する場合には）レジスタ名，およびそのメンバがどのように使用されるかを表わす記号です。ここで使われる記号は次のような意味を持っています。

R	プログラムから BIOS に渡す
W	BIOS からプログラムに返す
RW	プログラムから BIOS に渡し，BIOS が返す
Rx	プログラムから BIOS に渡す，BIOS は上位8ビットのみを使用する
×	使用しない（以前の値は保証しない）

当然のことながら，cmmd はすべて R になります。また stus は（使用される場合は）すべて W になります。それ以下は関数によって異なっています。

bios98com

機能 RS232C-BIOS を操作します。

形式 #include <bios98.h>
unsigned bios98com(COM_INFO * *cmf*);

プロトタイプ bios98.h

解説 **bios98com** は，受信データサイズの取得，データの送信，データの受信，コントローラへのコマンド出力，コントローラの状態の取得を行ないます。
構造体 **COM_INFO** は，bios98.h の中で次のように定義されています。

```
typedef struct {
   unsigned char  cmmd;      /* 機能コード          */
   unsigned char  stus;      /* 実行後の状態        */
   unsigned int   data_siz;  /* 受信データサイズ    */
   unsigned char  data;      /* 送受信データ        */
   unsigned char  data_inf;  /* 受信時データの状態  */
   unsigned char  ctrl_inf;  /* コントローラの状態  */
   unsigned char  port_inf;  /* システムポートの状態 */
} COM_INFO;
```

割り込みベクタテーブル(00000h～003FFh)内のタイマ割り込みと，RS-232C-BIOS で使用する割り込みベクタは，常に ROM 内の BIOS ルーチンを指しているとは限らないため，**bios98com** では，これらを利用可能な状態にしてからプログラムを実行しています。通常は，ユーザプログラムが終了した時点で，自動的に割り込みベクタを復帰させていますが，ユーザプログラムを常駐終了させたい場合には，下に示す変数をプログラム内で外部定義してください。

```
int  bs_com = 1;
```

bs_com を0以外にすると，タイマ割り込み(INT 0Ch)と RS-232C BIOS (INT 19h)の割り込みベクタを，元の状態に戻さないようになります。
MS-DOS がサポートする RS-232C インターフェースと混在して使用しないでください。また，MS-DOS の RS-232C デバイスドライバは必要としま

せん。

戻り値　各コマンドを実行した結果は16ビット中の下位8ビットに設定され返されます。また，パケット内のメンバstusにも格納されます。

■パケット内のデータの授受

COM_INFO		2	3	4	5	6
cmmd	AH	R	R	R	R	R
stus	AH	W	W	W	W	W
data_siz	CX	W	×	×	×	×
data	ALCH	×	R	W	×	×
data_inf	CL	×	×	W	×	×
ctrl_inf	ALCH	×	×	×	R	W
port_inf	CL	×	×	×	×	W

関連項目　**bios98com_ch2**, **bios98com_ch3**,
bios98com_init, **bios98com_init_ch2**, **bios98com_init_ch3**

例

```
#include <stdio.h>
#include <bios98.h>

#define ON  1
#define OFF 0

void com_init(unsigned char *buf)
{
   COM_INIT cint;
   cint.cmmd = 1;                  /* X パラメータつき */
   cint.transfer = 0x06;
   cint.mode_inf = 0x5E;
   cint.cmmd_inf = 0x37;
   cint.recv_buf = buf;
   cint.recv_siz = 256;
   cint.time_snd = 0x10;
   cint.time_rcv = 0x0F;
   bios98com_init(&cint);          /* RS-232C の初期化 */
   if (cint.stus != 0)
      printf("init_stus=%02X.¥n", cint.stus);
}

unsigned com_recvsize()
{
   COM_INFO cinf;
   cinf.cmmd = 2;
   bios98com(&cinf);               /* 受信データサイズの取得 */
   if (cinf.stus != 0)
      printf("recvsize_stus=%02X.¥n", cinf.stus);
   return cinf.data_siz;
}

void com_send(unsigned ch)
{
   COM_INFO cinf;
   cinf.cmmd = 3;
   cinf.data = ch;
   bios98com( &cinf );             /* データ送信 */
   if (cinf.stus != 0)
      printf("send_stus=%02X.¥n", cinf.stus);
}

unsigned com_recv(void)
{
   COM_INFO cinf;
     cinf.cmmd = 4;
     bios98com(&cinf);              /* データ受信 */
     if(cinf.stus != 0)
        printf("recv_stus=%02X.¥n", cinf.stus);
     return cinf.data;
}
```

```
void main()                 /* パソコン対パソコン通信テスト */
{
   unsigned char buf[300], sstr[80];
   int  sch, rch, send, i, n;
   com_init(buf);
   while (1) {
      gets(sstr);                 /* キーデータの入力 */
      if (sstr[0] == 0) {
         printf("受信を開始します。¥n");
         while ((rch = com_recv()) != 0)
            printf("%c", rch);    /* 受信データの表示 */
         printf("¥n");
      }
      else {
         printf("送信を開始します。¥n");
         n = strlen(sstr) + 1;
         for (i = 0; i < n; i++) {
            sch = sstr[i];
            do {
               while (com_recvsize() > 0) {
                  rch = com_recv();
                  switch (rch) {
                     case 0x13 : send = OFF;
                                 printf("¥n送信休止¥n");
                                 break;
                     case 0x11 : send = ON;
                                 printf("¥n送信再開¥n");
                                 break;
                     default   :      /* 送信中に受信したデータ */
                                 printf("%c", rch);
                                 break;
                  }
                  if (send == ON)  break;
               }
            } while (send == OFF);
            com_send(sch);        /* データの送信 */
         }
      }
   }
}
```

bios98com_ch2

機能　RS-232Cの2回線目用インターフェースです。

形式　#include <bios98.h>
unsigned bios98com_ch2(COM_INIT * *cmi*);

プロトタイプ　bios98.h

解説　拡張RS-232Cの2回線目用のインターフェースです。
詳細については **bios98com** を参照してください。

bios98com_ch3

機能　RS-232Cの3回線目用インターフェースです。

形式　#include <bios98.h>
unsigned bios98com_ch3(COM_INIT * *cmi*);

プロトタイプ　bios98.h

解説　拡張RS-232Cの3回線目用のインターフェースです。
詳細については **bios98com** を参照してください。

bios98com_init

機能　RS-232C を初期化します。

形式　#include <bios98.h>
unsigned bios98com_init(COM_INIT * *cmi*);

プロトタイプ　bios98.h

解説　**bios98com_init** は，RS-232C の初期化，制御を伴う初期化を行ないます。構造体 **COM_INIT** は，bios98.h の中で次のように定義されています。

```
typedef struct {
   unsigned char  cmmd;       /* 機能コード              */
   unsigned char  stus;       /* 実行後の状態            */
   unsigned char  transfer;   /* トランスファレート      */
   unsigned char  mode_inf;   /* モード指定              */
   unsigned char  cmmd_inf;   /* コマンド指定            */
   unsigned char *recv_buf;   /* 受信バッファ            */
   unsigned int   recv_siz;   /* 受信バッファサイズ      */
   unsigned char  time_snd;   /* 送信時のタイムアウト時間 */
   unsigned char  time_rcv;   /* 受信時のタイムアウト時間 */
} COM_INIT;
```

割り込みベクタテーブル(00000h～003FFh)内のタイマ割り込みと，RS-232C-BIOS で使用する割り込みベクタは，常に ROM 内の BIOS ルーチンを指しているとは限らないため，**bios98com_init** では，これらを利用可能な状態にしてからプログラムを実行しています。通常は，ユーザプログラムが終了した時点で，自動的に割り込みベクタを復帰させていますが，ユーザプログラムを常駐終了させたい場合には，下に示す変数をプログラム内で外部定義してください。

```
int  bs_com = 1;
```

bs_com を0以外にすると，タイマ割り込み(INT　0Ch)と RS-232C-BIOS (INT 19h)の割り込みベクタを，元の状態に戻さないようになります。MS-DOS がサポートする RS-232C インターフェースと混在して使用しないでください。また，MS-DOS の RS-232C デバイスドライバは必要としません。拡張 RS-232C (**bios98com_init_ch2**，**bios98com_init_ch3**) で使用

するとき，パケット内のトランスファレート transfer は使用しません（増設ボード上のディップスイッチで指定します)。

戻り値

各コマンドを実行した結果は16ビット中の下位8ビットに設定され返されます。また，パケット内のメンバ stus にも格納されます。

■パケット内のデータの授受

COM_INIT		0	1
cmmd	AH	R	R
stus	AH	W	W
transfer	AL	R	R
mode_inf	CH	R	R
cmmd_inf	CL	R	R
recv_buf	ESDI	R	R
recv_siz	DX	R	R
time_snd	BH	R	R
time_rcv	BL	R	R

可搬性

PC-9801シリーズでのみ動作します。

関連項目

bios98com, **bios98com_ch2**, **bios98com_ch3**, **bios98com_init_ch2**, **bios98com_init_ch3**

例

bios98com を参照してください。

bios98com_init_ch2

機能　2回線目の RS-232C を初期化します。

形式
```
#include <bios98.h>
unsigned bios98com_init_ch2(COM_INIT *cmi);
```

プロトタイプ　bios98.h

解説　拡張 RS-232C の2回線目用を初期化を行ないます。
詳細については，**bios98com_init** を参照してください。

戻り値　拡張 RS-232C 増設ボードを実装しないでこの関数を呼び出すと，0以外を返します。

bios98com_init_ch3

機能　3回線目の RS-232C を初期化します。

形式
```
#include <bios98.h>
unsigned bios98com_init_ch3(COM_INIT *cmi);
```

プロトタイプ　bios98.h

解説　拡張 RS-232C の3回線目用のインターフェースです。
詳細については **bios98com_init** を参照してください。

戻り値　拡張 RS-232C 増設ボードを実装しないでこの関数を呼び出すと，0以外を返します。

bios98disk

機能　ディスク BIOS を操作します。

形式　# include <bios98.h>

unsigned bios98disk(DISK_INFO * *dsk*);

プロトタイプ　bios98.h

解説　**bios98disk** は，フロッピィディスクを直接制御するためのルーチンです。パケット引数 *dsk* 内のメンバ cmmd により各種のコマンドを実行します。構造体 **DISK_INFO** は，bios98.h の中で次のように定義されています。

```
typedef struct {
   unsigned char cmmd;          /* DISKコマンド */
   unsigned char stus;          /* 実行後の状態 */
   unsigned char device_typ;    /* デバイスタイプ/ユニット番号 */
   unsigned int  data_siz;      /* データサイズ */
   unsigned char sector_siz;    /* セクタサイズ */
   unsigned char cylind_num;    /* シリンダ番号 */
   unsigned char head_num;      /* ヘッド番号 */
   unsigned char sector_num;    /* セクタ番号 */
   unsigned char *data_buf;     /* データ域 */
} DISK_INFO;
```

戻り値　　各コマンドを実行した結果は，16ビット中の下位8ビットに設定され返されます。また，パケット内のメンバstusにも格納されます。

■パケット内のデータの授受

1MB/640KB フロッピィディスクの場合：

DISK_INFO		1	2	3	4	6	7	8	9	10	11	12	14
cmmd	AH	R	R	R	R	R	R	R	R	R	R	R	R
stus	AH	W	W	W	W	W	W	W	W	W	W	W	W
device_typ	AL	R	R	R	R	R	R	R	R	R	R	R	R
data_siz	BX	R	R	R	×	×	R	×	×	R	R	R	R
sector_siz	CH	R	R	×	×	×	R	×	W	R	R	R	R
cylind_num	CL	R	R	R	×	×	R	×	RW	R	R	R	R
head_num	DH	R	R	×	×	×	R	×	RW	R	R	R	R
sector_num	DL	R	R	×	×	×	R	×	W	R	R	R	R
data_buf	ESBP	R	R	×	×	×	R	×	×	R	R	R	R

320**KB フロッピィディスクの場合**：

DISK_INFO		1	2	5	6	7	8	13
cmmd	AH	R	R	R	R	R	R	R
stus	AH	W	W	W	W	W	W	W
device_typ	AL	R	R	R	R	R	R	R
data_siz	BX	R	R	×	×	R	×	×
sector_siz	CH	R	R	×	×	R	×	×
cylind_num	CL	R	R	×	×	R	×	×
head_num	DH	R	R	×	×	R	×	×
sector_num	DL	R	R	×	×	R	×	×
data_buf	ESBP	R	R	×	×	R	×	×

機能コード（DISK コマンド）は以下のような意味を持っています。

番号	コマンド	コード 1MB/640Kb	320Kb
1	READ DATA	0xT6	0x06
2	WRITE DATA	0xT5	0x05
3	SEEK	0xU0	
4	RECALIBRATE	0xV7	
5	FORMAT DRIVE		0x0C
6	INITIALIZE	0x03	0x03
7	VERIFY	0xT1	0x01
8	SENSE	0x04	0x04
9	READ ID	0xWA	
10	WRITE DELETED DATA	0xT9	
11	READ DELETED DATA	0xTC	
12	READ DIAGNOSTIC	0xW2	
13	SET OPERATION MODE		0x0E
14	FORMAT TRACK	0xWD	

上の表のコード使われているサブコマンド（上位4ビット）は以下のような値と意味を持っています。

上位4ビット				
T	MT	MF	r	SK
U	0	0	r	1
V	0	0	r	0
W	0	MF	r	SK

ビット	説明
MT	トラック指定
MF	密度指定
r	再試行指定
SK	シーク指定

例

```
#include <bios98.h>
void read_sec(unsigned char *buf, int c, int h, int s)
{
   DISK_INFO dinf;
   dinf.cmmd        = 0xDA;
   dinf.device_typ = 0x90;    /* 標準フロッピィドライブ0 */
   dinf.sector_siz = -1;
   dinf.cylind_num = c;
   dinf.head_num    = h;
   dinf.sector_num = -1;
   bios98disk(&dinf);         /* セクタサイズの取得 */
   dinf.cmmd = 0xD6;
   dinf.data_siz    = 1024;
   dinf.sector_num = s;
   dinf.data_buf    = buf;
   bios98disk(&dinf);         /* データの読み出し */
   printf("stus=%02X, CY=%d, HD=%d, SN=%d, SS=%d.¥n",
           dinf.stus, dinf.cylind_num, dinf.head_num,
           dinf.sector_num, dinf.sector_siz);
}

void main()
{
   char buf[2048];
   int  i, j, k;

   for (i = 0; i <= 76; i++) {
      for (j = 0; j <= 1; j++) {
         for (k = 1; k <= 8; k++) {
            read_sec(buf, i, j, k);
            /* ... */
         }
      }
   }
}
```

bios98equip

機能　システム情報を取得します。

形式

```
#include <bios98.h>
unsigned bios98equip(void);
```

プロトタイプ　bios98.h

解説　**bios98equip** は，現在システムに接続されている機器情報を返します。

注記：この関数は BIOS インターフェースではありません。

戻り値　16ビット中に下記のように情報が返されます。

ビット	名前	意味	
15	GPIB ボード	0：なし	1：あり
14	拡張 RS-232C ボード	0：なし	1：あり
13	サウンドボード	0：なし	1：あり
12	未使用		
11-08	接続ドライブ数	0～15	
07-04	クロックモード	0：5MHz	3：12MHz
		1：8MHz	4：16MHz
		2：10MHz	
03-01	CPU タイプ	0：8086	2：80286
		1：V30	3：80386
00	キーボードタイプ	0：従来型	1：新型

例

```
#include <bios98.h>

void main()
{
   unsigned flag;

   flag = bios98equip();
   if (flag & 0x8000)
      printf("GPIBボードが接続されています。¥n");
   if (flag & 0x4000)
      printf("拡張RS-232Cボードが接続されています。¥n");
   if (flag & 0x2000)
      printf("サウンドボードが接続されています。¥n");
   printf("ドライブは %d 台 接続されています。¥n",
          (flag >> 8) & 0x0F );
   switch (( flag >> 8 ) & 0x0F ) {
      case 0 : printf("CPUクロックは5MHzです。¥n");
               break;
      case 1 : printf("CPUクロックは8MHzです。¥n");
               break;
      case 2 : printf("CPUクロックは10MHzです。¥n");
               break;
      case 3 : printf("CPUクロックは12MHzです。¥n");
               break;
      case 4 : printf("CPUクロックは16MHzです。¥n");
               break;
   }
   switch (( flag >> 1 ) & 0x0007 ) {
      case 0 : printf("CPUタイプは8086です。¥n");
               break;
      case 1 : printf("CPUタイプはV30です。¥n");
               break;
      case 2 : printf("CPUタイプは80286です。¥n");
               break;
      case 3 : printf("CPUタイプは80386です。¥n");
               break;
   }
   if (flag & 0x0001)
      printf("キーボードは新型です。¥n");
   else
      printf("キーボードは従来型です。¥n");
}
```

bios98harddisk

機能　ハードディスクを操作します。

形式　#include <bios98.h>

unsigned bios98harddisk(HARDDISK_INFO * *hdk*);

プロトタイプ　bios98.h

解説　**bios98harddisk** は，ハードディスクを直接制御するためのインターフェースです。

パケット引数 *hdk* 内のメンバ cmmd により各種のコマンドを実行します。

構造体 **HARDDISK_INFO** は，bios98.h の中で次のように定義されています。

```
typedef struct {
   unsigned char cmmd;         /* DISKコマンド */
   unsigned char stus;         /* 実行後の状態 */
   unsigned char device_typ;   /* デバイスタイプ/ユニット番号 */
   unsigned int  data_siz;     /* データサイズ */
   unsigned int  cylind_num;   /* シリンダ番号 */
   unsigned char head_num;     /* ヘッド番号 */
   unsigned char sector_num;   /* セクタ番号 */
   unsigned char *data_buf;    /* データ域 */
} HARDDISK_INFO;
```

戻り値　各コマンドを実行した結果は，16ビット中の下位8ビットに設定され返されます。また，パケット内のメンバ stus にも格納されます。

■パケット内のデータの授受

HARDDISK_INFO		1	2	4	5	6	7	8	14	15	16	17
cmmd	AH	R	R	R	R	R	R	R	R	R	R	R
stus	AH	W	W	W	W	W	W	W	W	W	W	W
device_typ	AL	R	R	R	R	R	R	R	R	R	R	R
data_siz	BXBH	R	R	×	Rx	×	R	×	Rx	R	R	×
cylind_num	CX	R	R	×	R	×	R	×	R	R	R	×
head_num	DH	R	R	×	R	×	R	×	R	R	R	×
sector_num	DL	R	R	×	R	×	R	×	R	R	R	×
data_buf	ESBP	R	R	×	×	×	R	×	×	R	R	×

機能コード（DISK コマンド）は以下のような意味を持っています。

番号	コマンド	コード
1	READ DATA	0xX6
2	WRITE DATA	0xX5
4	RECALIBRATE	0xX7
5	FORMAT DRIVE	0xYD
6	INITIALIZE	0x03
7	VERIFY	0xX1
8	SENSE	0x04
14	FORMAT TRACK	0xZD
15	ASSIGN ALTERNATE TRACK	0x08
16	FORMAT BAD TRACK	0x0B
17	RETRACT	0xXF

上の表のコード使われているサブコマンド（上位4ビット）は以下のような値と意味を持っています。

上位4ビット				
X	×	×	r	×
Y	1	×	r	×
Z	0	×	r	×

ビット	説明
r	再試行指定
×	意味なし

bios98key

機能　キーボードインターフェースです。

形式　# include <bios98.h>

void bios98key(KEY_INFO * *key*);

プロトタイプ　bios98.h

解説　**bios98key** は，入力待ちのキーデータの読み出し，入力待ちなしキーデータの読み出し，シフトキーの押下チェック，キーボードインターフェースの初期化，グループわけされたキー配列のキー入力状態のセンスなどを行ないます。

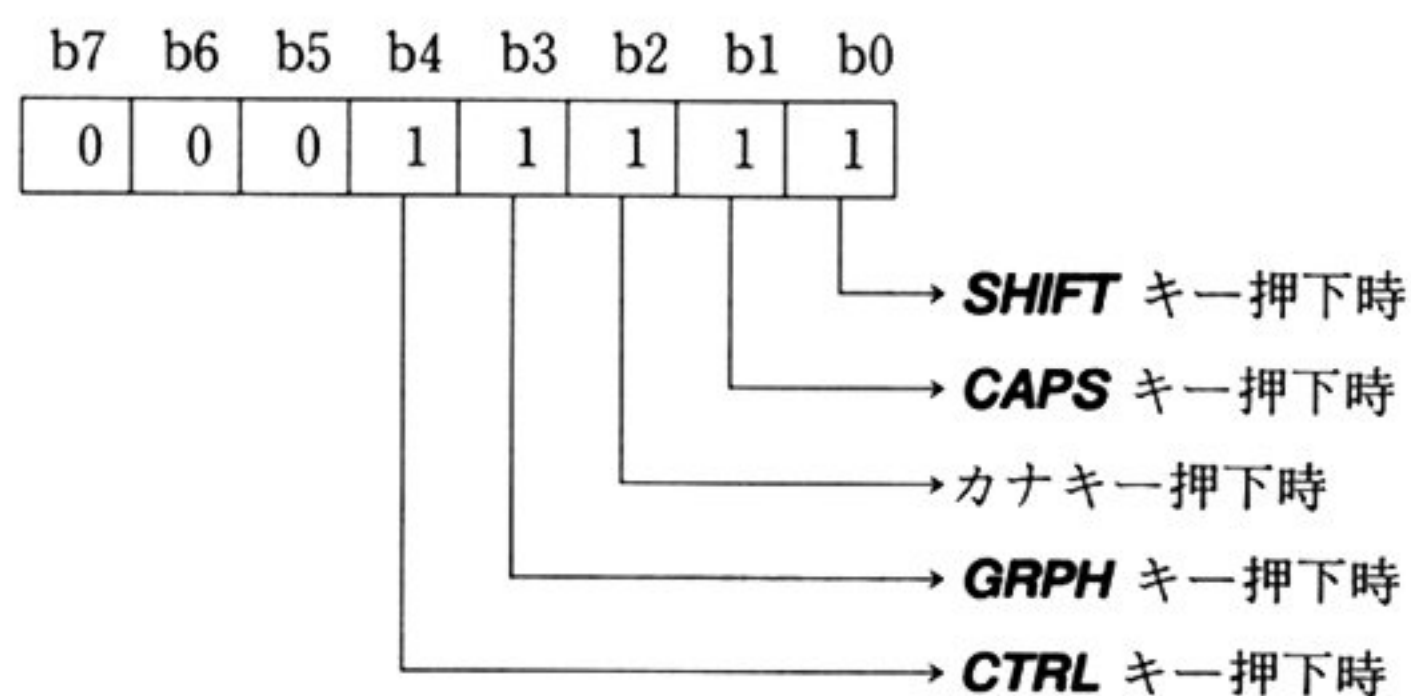
構造体 **KEY_INFO** は，bios98.h の中で以下のように定義されています。

```
typedef struct {
   unsigned char  cmmd;          /* 機能コード */
   unsigned char  stus;          /* 実行後の状態 */
   unsigned char  scan_code;     /* スキャンコード */
   unsigned char  inter_code;    /* 内部コード */
   unsigned char  shift_code;    /* シフトコード */
   unsigned char  group_num;     /* グループ番号 */
   unsigned char  group_stus;    /* グループビット情報 */
} KEY_INFO;
```

この関数の実行中は，***STOP*** キーは完全に無視されるようになっています。

戻り値　機能コードが0のとき，入力されたキーコードを返します。16ビット中の上位8ビットはスキャンコードで16ビット中の下位8ビットは内部コードです。また，パケット内のメンバにも値が格納されます。

機能コードが1のとき，キーが解放状態のときは0を返し，キーが押下状態のときはそのキーコードを返します。16ビット中の上位8ビットはスキャンコードで16ビット中の下位8ビットは内部コードです。また，パケット内のメンバにも格納されます。

機能コードが2のとき，シフトキー押下情報を返します。16ビット中の下位8ビットに下記のように情報を返します。また，パケット内のメンバにも格納されます。

機能コードが3のとき，戻り値は意味を持ちません。

機能コードが4のとき，変数 group_num のキーコードグループに対応するビット情報を返します。16ビット中の下位8ビットはキーの状態を示しています。また，パケット内のメンバにも格納されます。

■パケット内のデータの授受

KEY_INFO		0	1	2	3	4
cmmd	AH	R	R	R	R	R
stus	BH	×	W	×	×	×
scan_code	AH	W	W	×	×	×
inter_code	AL	W	W	×	×	×
shift_code	AL	×	×	W	×	×
group_num	AL	×	×	×	×	R
group_stus	AH	×	×	×	×	W

例

```
#include <bios98.h>

void main()
{
   int i;
   unsigned key, sft;
   KEY_INFO kinf;

   kinf.cmmd = 3;
   bios98key(&kinf);              /* キーボードの初期化 */
   printf("キーを押してください。¥n");
   kinf.cmmd = 0;
   key = bios98key(&kinf);        /* キー入力 */
   kinf.cmmd = 2;
   sft = bios98key(&kinf);        /* シフトキーのセンス */
   printf("スキャンコードは %02X, 内部コードは %02X です。¥n",
           key / 256, key % 256);
   printf("シフトキー状態は %02X です。¥n", sft);
   do {
      printf("キーセンス中, ");
      kinf.cmmd = 1;
      key = bios98key(&kinf); /* キー入力 */
   } while (key == 0);
   printf("スキャンコードは %02X, 内部コードは %02X です。¥n",
           key / 256, key % 256);
   i = 0;
   do {
      kinf.cmmd = 4;
      kinf.group_num = i;
      key = bios98key(&kinf); /* キーグループセンス */
      printf("キーグループ%02X, 対応ビット%02X¥n", i, key);
      i++;
   } while (i < 16);
}
```

bios98memory

機能　実装メモリのサイズを調べます。

形式　# include <bios98.h>
unsigned bios98memory(void);

プロトタイプ　bios98.h

解説　**bios98memory** は，現在使用しているシステムのメインメモリサイズを返します。この関数は，メモリスイッチ（**bios98msw** を参照）のメモリサイズをチェックするのではなく，実メモリサイズをチェックします。

注記：この関数は BIOS インターフェースではありません。

戻り値　128，256，384，512，640のいずれかを返します。単位は K バイトです。

例

```
#include <bios98.h>
void main()
{
   printf("システムのメモリは %dKbです。¥n", bios98memory());
}
```

bios98mouse

機能　マウスの制御を行ないます。

形式　# include <bios98.h>
void bios98mouse(MOUSE_INFO * *msf*);

プロトタイプ　bios98.h

解説　**bios98mouse**は，カーソル位置の取得，カーソル位置の設定，左ボタン押下情報の取得，左ボタン解放情報の取得，右ボタン押下情報の取得，右ボタン解放情報の取得，マウス移動距離情報の取得などのマウスの動作に関するコマンドを実行します。また，環境の初期化，カーソル表示，カーソル消去などはどちらの関数で利用してもかまいません。
構造体**MOUSE_INFO**は，bios98.hの中で以下のように定義されています。

```
typedef struct {
   unsigned char cmmd;         /* 機能コード              */
   unsigned char stus;         /* 実行後の状態            */
   char          lft_buton;    /* 左ボタンの状態          */
   char          rgt_buton;    /* 右ボタンの状態          */
   unsigned int  lft_count;    /* 左ボタンの押下/解放回数  */
   unsigned int  rgt_count;    /* 右ボタンの押下/解放回数  */
   unsigned int  abs_h_crs;    /* カーソルの水平軸座標     */
   unsigned int  abs_v_crs;    /* カーソルの垂直軸座標     */
   int           rel_h_crs;    /* カーソルの相対水平軸座標 */
   int           rel_v_crs;    /* カーソルの相対水直軸座標 */
} MOUSE_INFO;
```

注意：この関数は，日本電気が提供しているマウスドライバを利用しています。したがって使用する場合は，必らずCONFIG.SYSにマウスドライバ（MOUSE.SYS）を組み込むこんでおいてください。

戻り値　　ありません。

■パケット内のデータの授受

MOUSE_INFO		0	1	2	3	4	5	6	7	8	11
cmmd	AH	R	R	R	R	R	R	R	R	R	R
stus	AX	W	×	×	×	×	×	×	×	×	×
lft_buton	AX	×	×	×	W	×	W	W	×	×	×
rgt_buton	BX	×	×	×	W	×	×	×	W	W	×
lft_count	BX	×	×	×	×	×	W	W	×	×	×
rgt_count	BX	×	×	×	×	×	×	×	W	W	×
as_h_crs	CX	×	×	×	W	R	W	W	W	W	×
abs_v_crs	DX	×	×	×	W	R	W	W	W	W	×
rel_h_crs	CX	×	×	×	×	×	×	×	×	×	W
rel_v_crs	DX	×	×	×	×	×	×	×	×	×	W

関連項目　　**bios98mouse_init**

例

```
#include <bios98.h>
static unsigned char  mouse_patt[] = { ........ };

void mousefun(int stus, int lft_buton, int rgt_buton,
              unsigned abs_h_crs, unsigned abs_v_crs)  {
   switch (stus) {
      case  1 : /* ... */
      case  2 : /* ... */
      case  4 : /* ... */
      case  8 : /* ... */
      case 16 : /* ... */
   }
}

void main()
{
   MOUSE_INIT mint;
   MOUSE_INFO minf;
   mint.cmmd = 0;
   bios98mouse_init(&mint);        /* マウス環境の初期化 */
   mint.cmmd = 18;
   mint.plane_num = 1;
   bios98mouse_init(&mint);        /* マウス描写画面の設定 */
   mint.cmmd = 9;
   mint.crs_h_bas = 0; mint.crs_v_bas = 0;
   mint.crs_pat_p = mouse_patt;
   bios98mouse_init(&mint);        /* カーソル形状の設定 */
   mint.cmmd = 15;
   mint.md_h_sca = 2; mint.md_v_sca = 2;
   bios98mouse_init(&mint);        /* ミッキー／ドット比の設定 */
   mint.cmmd = 12;
   mint.call_cond = 0x1F;
   mint.mouse_fun = mousefunc;
   bios98mouse_init(&mint);        /* 関数登録 */
   minf.cmmd = 4;
   minf.abs_h_crs = 300; minf.abs_v_crs = 200;
   bios98mouse(&minf);             /* マウスカーソル位置設定 */
   mint.cmmd = 1;
   bios98mouse_init(&mint);        /* マウスカーソルの表示 */
   minf.cmmd = 5;
   bios98mouse(&minf);             /* 左ボタン押下の取得 */
   printf("L=%d,LC=%d,H=%d,V=%d¥n",
           minf.lft_buton, minf.lft_count,
           minf.abs_h_crs, minf.abs_v_crs);
      /* ... */
   minf.cmmd = 8;
   bios98mouse(&minf);         /* 右ボタン解放の取得 */
   printf("R=%d, RC=%d, H=%d, V=%d¥n",
           minf.rgt_buton, minf.rgt_count,
           minf.abs_h_crs, minf.abs_v_crs);
      /* ... */
   mint.cmmd = 2;
   bios98mouse_init(&mint);    /* マウスカーソルの消去 */
   mint.cmmd = 0;
   bios98mouse_init(&mint);    /* マウス環境の初期化 */
}
```

bios98mouse_init

機能　マウスの初期化を行ないます。

形式
```
#include <bios98.h>
void bios98mouse_init(MOUSE_INIT * msi);
```

プロトタイプ　bios98.h

解説　**bios98mouse_init**は，カーソル表示画面の設定，水平軸方向のカーソル移動範囲の設定，垂直軸方向のカーソル移動範囲の設定，ミッキー/ドット比の設定，カーソル形状の設定，ユーザ定義の割り込みルーチンを登録などの初期化に関するコマンドを実行します。

構造体**MOUSE_INIT**は，bios98.hの中で以下のように定義されています。

```
typedef struct {
   unsigned char  cmmd;        /* 機能コード                 */
   unsigned char  stus;        /* 実行後の状態               */
   unsigned int   plane_num;   /* カーソルの表示画面         */
   unsigned int   min_h_pos;   /* カーソル水平軸の下限界     */
   unsigned int   max_h_pos;   /* カーソル水平軸の上限界     */
   unsigned int   min_v_pos;   /* カーソル垂直軸の下限界     */
   unsigned int   max_v_pos;   /* カーソル垂直軸の上限界     */
   unsigned int   md_h_scal;   /* 水平軸のミッキー/ドット比  */
   unsigned int   md_v_scal;   /* 垂直軸のミッキー/ドット比  */
   unsigned int   crs_h_bas;   /* マウスカーソル水平軸の基点 */
   unsigned int   crs_v_bas;   /* マウスカーソル垂直軸の基点 */
   unsigned char *crs_pat_p;   /* マウスカーソルの形状を指す */
   unsigned int   call_cond;   /* 割り込み条件               */
   void       (*mouse_fun)();  /* 割り込み時に起動する関数   */
} MOUSE_INIT;
```

機能コード12の“ユーザ定義のサブルーチンの登録”において，ユーザ定義の割り込みハンドラを呼び出す形式は下記の通りです。

```
void mouse_fun(int      stus,        /* 1,2,4,8,16:原因 */
               int      lft_buton,   /* 0:解放, -1:押下 */
               int      rgt_buton,   /* 0:解放, -1:押下 */
               unsigned abs_h_crs,   /* 0～639 */
               unsigned abs_v_crs);  /* N:0～199, H:0～399 */
```

この中で，Nはカラーモード，Hは高分解カラーモードを表わします。
なお，ユーザ定義の関数内で実行できる処理は下記のような制約を受けます。

■ DOSシステムコールを発行してはいけません。DOSシステムコールを処理中でなければかまいません(C標準ライブラリ，C非標準ライブラリの関数はこれらを使用しているものがあるので充分注意してください)。

■ ROM-BIOSコールはできるだけ使用を避けてください。

■タイニィ，スモール，ミディアムモデルでは自動（auto）の配列や構造体は使用してはいけません。これは，割り込みが発生するタイミングは常にユーザ定義のプログラムが稼働しているときだけではなく，システム側のタスク（最小単位の意味ある処理工程）が起動しているときでも発生するからです。
割り込みがシステム側のタスク内で発生したときは，スタックセグメント（SS）がシステム側の領域を指しているため不具合が生じます。タイニィ，スモール，ミディアムモデルではDS,ES,SSセグメントレジスタは常に同一領域を示していなければならないという規則があります。DS,ESセグメントレジスタに関してはユーザ定義の割り込みハンドラに制御を渡す前に，ユーザ定義のプログラム側の領域を指すようにしてあります。静的（static）変数，単独の自動（auto）変数は利用できます。
なお，コンパクト，ラージ，ヒュージモデルではこうした制約なしで利用できます。

■ **setjmp**，**longjmp**は使用できません。

■割り込みハンドラが使用するスタックは，システム側のスタックか利用者側のスタックか解らないので，関数の呼び出しのネストを深くしないようにしてください。
また，コンパイルの際には，スタックオーバーフローチェックのオプション(-N)はつけないでください。

この機能コードの使用に関して，利用者が作成したプログラムを終了させるときには，再度マウス環境を初期化しなければなりません。初期化しておかないと，プログラム終了後，マウスに触れただけでシステムが暴走するので注意して使ってください。

注意：この関数は，日本電気が提供しているマウスドライバを利用しています。したがって使用する場合は，必らずCONFIG.SYSにマウスドライバ（MOUSE.SYS）を組み込むこんでおいてください。

戻り値　　ありません。

■パケット内のデータの授受

MOUSE_INIT		0	1	2	9	12	15	16	17	18
cmmd	AH	R	R	R	R	R	R	R	R	R
stus	AX	W	×	×	×	×	×	×	×	×
plane_num	BX	×	×	×	×	×	×	×	×	R
min_h_pos	CX	×	×	×	×	×	×	R	×	×
max_h_pos	DX	×	×	×	×	×	×	R	×	×
min_v_pos	CX	×	×	×	×	×	×	×	R	×
max_v_pos	DX	×	×	×	×	×	×	×	R	×
md_h_scal	CX	×	×	×	×	×	R	×	×	×
md_v_scal	DX	×	×	×	×	×	R	×	×	×
crs_h_bas	BX	×	×	×	R	×	×	×	×	×
crs_v_bas	CX	×	×	×	R	×	×	×	×	×
crs_pat_p	ESDX	×	×	×	R	×	×	×	×	×
call_cond	CX	×	×	×	×	R	×	×	×	×
mouse_fun	ESDX	×	×	×	×	R	×	×	×	×

関連項目　　**bios98mouse**

例　　**bios98mouse**を参照してください。

bios98msw

機能　メモリスイッチ情報を得ます。

形式　#include <bios98.h>
unsigned bios98msw(MSW_INFO * *msw*) ;

プロトタイプ　bios98.h

解説　**bios98msw** は, PC-9801シリーズ上のメモリスイッチ情報の読み出し, 更新を行ないます。更新の際には, 本体のディップスイッチSW2の5がONになっている必要があります。SW2の5がOFFになっていると, 本体の電源を再投入したときにメモリスイッチが初期化されてしまうため, メモリスイッチへの書き込みを行なっても情報が失われてしまいます。
引数sw_numには1～6の整数を指定し, それぞれメモリスイッチのSW1～SW6に対応します。
構造体 **MSW_INFO** は, bios98.hで以下のように定義されています。

```
typedef struct {
   unsigned char  cmmd;     /* 機能コード          */
   unsigned char  stus;     /* 実行後の状態        */
   int            sw_num;   /* メモリスイッチ番号  */
   unsigned char  value;    /* メモリスイッチの内容 */
} MSW_INFO;
```

機能コード1は指定したメモリスイッチの読み出し, 機能コード2は指定したメモリスイッチの書き出しになります。

注記：この関数はBIOSインターフェースではありません。

戻り値　機能コードが1のとき，指定されたメモリスイッチ情報を返します。また，パケット内のメンバにも格納されます。
機能コードが2のとき，引数 value の値を返します。sw_num の値が不正なときは-1を返します。

■パケット内のデータの授受

MSW_INFO	1	2
cmmd	R	R
stus	×	×
sw_num	R	R
value	W	R

例

```
#include <bios98.h>
void main()
{
   unsigned stus;
   int  i;
   MSW_INFO  minf;

   for (i = 1; i <= 6; i++) {
      minf.cmmd = 1;
      minf.sw_num = i;
      stus = bios98msw(&minf);          /* 読み出し */
      printf("SW%d = %02X.¥n", i, stus);
   }
   minf.cmmd = 2;
   minf.sw_num = 6;
   minf.value = 0x14;
   stus = bios98msw(&minf);             /* 書き込み */
   printf("SW6 = %02X.¥n", stus);
}
```

bios98print

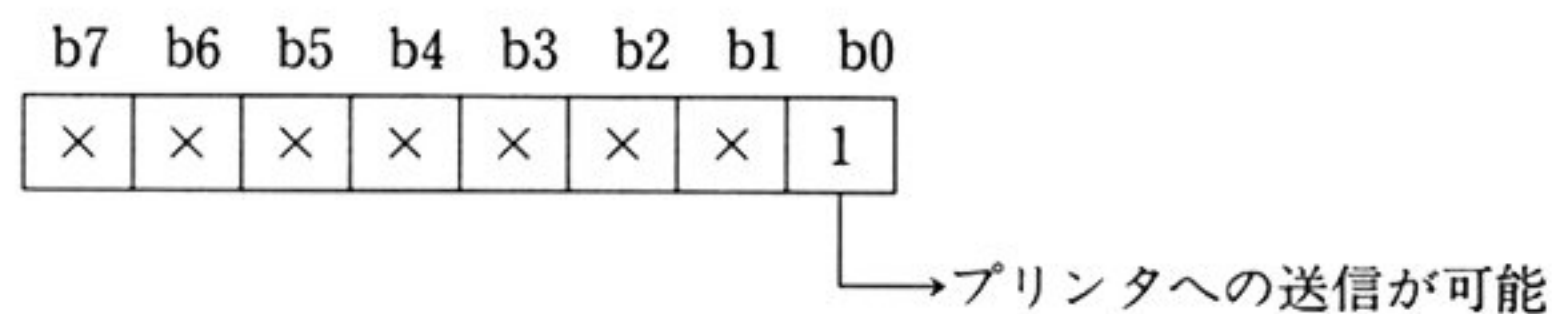
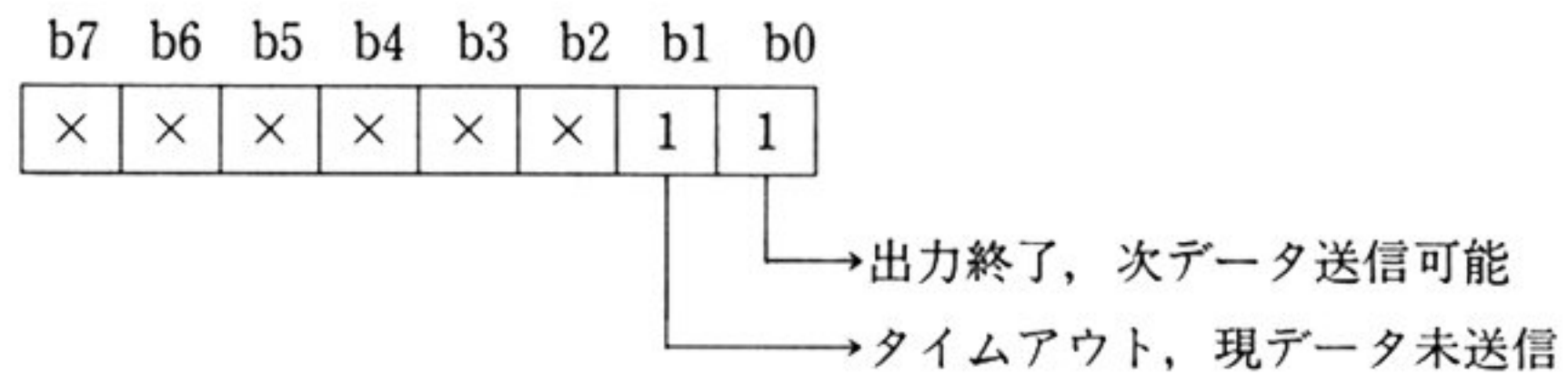
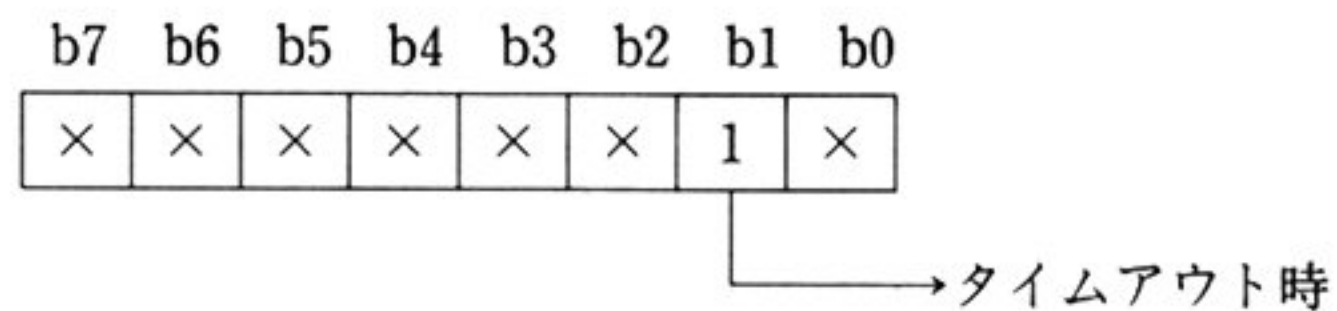
機能　プリンタインターフェースです。

形式　# include <bios98.h>
unsigned bios98print(PRINT_INFO * *prt*)；

プロトタイプ　bios98.h

解説　**bios98print** は，プリンタコントローラの初期化，プリンタステータスの取得，1文字単位の出力，文字列単位の出力に使用します。
構造体 **PRINT_INFO** は，bios98.h の中で以下のように定義されています。

```
typedef struct {
   unsigned char  cmmd;       /* 機能コード             */
   unsigned char  stus;       /* 実行後の状態           */
   unsigned char  chr;        /* 出力文字               */
   unsigned char *string;     /* 出力文字列を指すポインタ */
   unsigned char *string;     /* 出力サイズ             */
} PRINT_INFO;
```

戻り値　機能コードが16および18のときは，16ビット中の下位1ビットに，プリンタに対して送信可能かどうかの状態を返します。また，パケット内のメンバにも格納されます。

機能コードが17のときは，16ビット中の下位2ビットにプリンタへの送信状態を返します。また，パケット内のメンバにも格納されます。

機能コードが48のときは，ビットb1にタイムアウト情報を返し，正常終了のときは，パケット内のメンバstringおよびメンバsizeは不定となります。タイムアウト時にはメンバstringに未送信のデータの先頭アドレスを返し，メンバsizeに未送信のデータ長を返します。

■パケット内のデータの授受

PRINT_INFO		16	17	18	48
cmmd	AH	R	R	R	R
stus	AH	W	W	W	W
chr	AL	×	R	×	×
string	ESBX	×	×	×	RW
size	CX	×	×	×	RW

例

```
#include <bios98.h>
void main()
{
   unsigned   status;
   PRINT_INFO pinf;

   pinf.cmmd = 16;
   status = bios98print(&pinf);
                          /* プリンタコントローラの初期化 */
   if ( ! (status & 0x01)) {
      printf("プリンタの準備をしてください。¥n");
      pinf.cmmd = 18;
      status = bios98print( &pinf);
                              /* プリンタステータスの取得 */
   }
   do {
      pinf.cmmd = 17;
      pinf.chr = 'A';
      status = bios98print(&pinf); /* 1文字出力 */
      if ( ! (status & 0x01))
         printf("プリンタの電源を入れてください。¥n");
      if (status & 0x02)
         printf("用紙切れ 準備中 ハードエラー。¥n");
   } while (status != 0x01);
   pinf.cmmd = 48;
   pinf.string = "ABCDE";
   pinf.size = 5;
   status = bios98print(&pinf); /* 文字列出力 */
   if (status & 0x02) {
      printf("タイムアウトが発生しました。¥n");
      printf("%sの文字列が%dバイト分未送信です。¥n",
               pinf.string, pinf.size);
   }
}
```

bios98stoptimer

機能　割り込みハンドラを切り放します。

形式　void bios98stoptimer(void);

プロトタイプ　bios98.h

解説　**bios98stoptimer** は，ユーザプログラムを終了する際に，**bios98timer** によってインストールされた割り込みハンドラを強制的に切り放すときに使用します。
親プロセスや子プロセスで同時にタイマ割り込みを使用した場合，プログラムの動作は保証されません。
タイマ割り込みハンドラについては，**bios98timer** の解説を参照してください。

戻り値　ありません。

関連項目　**bios98timer**

例　**bios98timer** を参照してください。

bios98time

機能　日付時刻の読み出しおよび設定を行ないます。

形式　#include　<bios98.h>
void bios98time(TIME_INFO * *tim*);

プロトタイプ　bios98.h

解説　**bios98time** は日付時刻の読み出し，日付時刻の設定を行ないます。
構造体 **TIME_INFO** は，bios98.h の中で以下のように定義されています。

```
typedef struct {
   unsigned char cmmd;          /* 機能コード */
   unsigned char stus;          /* 実行後の状態 */
   unsigned char year;          /* 年 BCDコード 00～99 */
   unsigned char month_week;    /* 月，曜日 2進数 1～12，0～6 */
   unsigned char day;           /* 日 BCDコード 01～31 */
   unsigned char hour;          /* 時 BCDコード 00～23 */
   unsigned char minute;        /* 分 BCDコード 00～59 */
   unsigned char second;        /* 秒 BCDコード 00～59 */
} TIME_INFO;
```

戻り値　ありません。

■パケット内のデータの授受

TIME_INFO		0	1
cmmd	AH	R	R
stus		×	×
year		W	R
month_week		W	R
day		W	R
hour		W	R
minute		W	R
second		W	R

例

```
#include <bios98.h>
#define  BCDtoB(x)   ( (x / 16) * 10 + (x % 16) )
#define  BtoBCD(x)   ( (x / 10) * 16 + (x % 10) )
#define  GETupr(x)   ( x / 16 )
#define  GETlwr(x)   ( x % 16 )
static char *toJPN[]
            = { "日", "月", "火", "水", "木", "金", "土" };

void main()
{
   TIME_INFO  tinf;
   int        value;

   tinf.cmmd = 0;
   bios98time( &tinf);          /* 現日付時刻の取得 */
   printf("現日付は%d年%d月%d日 (%s) %d時%d分%d秒¥n",
          BCDtoB(tinf.year), GETupr(tinf.month_week),
          BCDtoB(tinf.day),  toJPN[GETlwr(tinf.month_week)],
          BCDtoB(tinf.hour), BCDtoB(tinf.minute),
          BCDtoB(tinf.second) );
   if (BCDtoB(tinf.second) < 30)
      tinf.second = BtoBCD(0);
   else {
      tinf.second = BtoBCD(0);
      value = BCDtoB(tinf.minute) + 1;
      if (value < 60)
         tinf.minute = BtoBCD(value);
      else {
         tinf.minute = BtoBCD(0);
         value = BCDtoB(tinf.hour) + 1;
         if (value < 24)
            tinf.hour = BtoBCD(value);
         else {
            tinf.hour = BtoBCD(0);
            value = BCDtoB(tinf.day) + 1;
         }
      }
   }
   tinf.cmmd = 1;
   bios98time(&tinf);          /* 日付時刻の設定 */
   printf("秒針を補正しました。¥n");
}
```

bios98timer

機能 インターバルタイマーインターフェース

形式 void bios98timer(unsigned *interval_time*, void (* *timer_func*)());

関連関数 void bios98stoptimer(void);

プロトタイプ bios98.h

解説 **bios98timer** は，インターバルタイマの設定と割り込みルーチンの設定を行ないます。インターバル時間 *interval_time* は，10msec 単位で指定してください。指定時間後に *timer_func* を1回だけ呼び出します。連続して割り込みを発生させるときは，*timer_func* 内で **bios98timer** を呼び出し，自分自身を再度登録しなおします。
ユーザ定義の *timer_func* の呼び出し形式は次のようになります。

```
void timer_func(void);
```

ユーザ定義関数内で実行できる処理は，以下のような制約を受けます。

- MS-DOS システムコールを発行してはいけません。MS-DOS システムコールを処理中でなければかまいません(C 標準ライブラリ，C 非標準ライブラリの関数はこれらを使用しているものがあるので充分注意してください)。
- ROM-BIOS コールはできるだけ使用を避けてください。
- タイニィ，スモール，ミディアムモデルでは自動属性（auto）の配列や構造体は使用してはいけません。これは，割り込みが発生するタイミングは，ユーザ定義のプログラムが稼働しているときだけではなく，システム側のタスク（最小単位の意味ある処理工程）が起動しているときでも発生するからです。
割り込みがシステム側のタスク内で発生したときは，スタックセグメント（SS）がシステム側の領域を指しているため不具合が生じます。

タイニィ，スモール，ミディアムモデルではDS,ES,SSセグメントレジスタは常に同一領域を示していなければならないという規則があります。DS,ESセグメントレジスタに関してはユーザ定義の割り込みハンドラに制御を渡す前に，ユーザ定義のプログラム側の領域を指すようになっています。静的（static）変数，単独の自動（auto）変数は利用できます。

なお，コンパクト，ラージ，ヒュージモデルではこうした制約なしで利用できます。

■ **setjmp**，**longjmp** は使用できません。

■割り込みハンドラが使用するスタックは，システム側のものかユーザ側のものかは解らないので，関数の呼び出しネストを深くしないようにしてください。

また，コンパイルの際には，スタックオーバーフローチェックのオプション(-N)はつけないでください。

ユーザ定義のプログラムを終了する際に，割り込みハンドラを強制的に切り放すには，**bios98stoptimer** を使用します。

戻り値　　ありません。

関連項目　　**bios98stoptimer**

例

```
#include <bios98.h>
static int  clock;
static int  delay_on;

void timefunc()
{
   clock++;                          /* 1秒カウンタ */
   bios98timer(100, timefunc);       /* 再度，1秒後に割込発生 */
}

void delayfunc()
{
   delay_on++;
}

void delay(unsigned milliseconds)
{
   delay_on = 0;
   bios98timer( milliseconds / 10, delayfunc);
   while(1)
     if (delay_on > 0) break;
}

void main()
{
   clock = 0;
   bios98timer(100, timefunc);       /* 1秒後に割込発生 */
   while (1) {
      printf("%u,", clock);
      if (clock > 1000) break;
  }
  bios98stoptimer();                 /* 割り込み停止 */
  /* ... */
  delay(10000);                      /* 10秒間休み */
  /* ... */
}
```

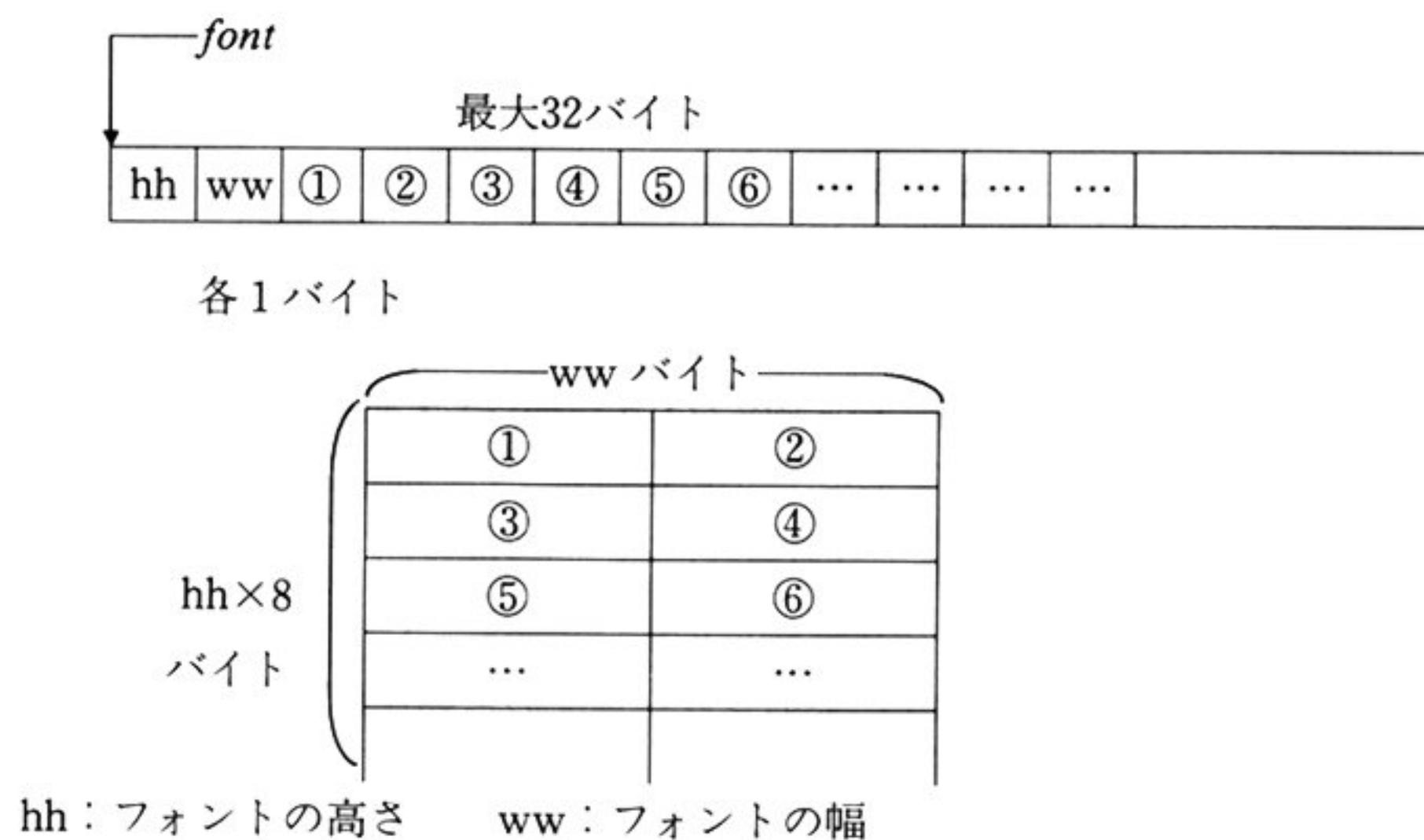
getfont

機能　ROM のフォントパターンを読み出します。

形式　void getfont(unsigned int *code*, *unsigned char * font*)；

プロトタイプ　dos.h

機能説明　**getfont** は，マシンの ROM に組み込まれているフォントパターンを読み出します。引数 *code* で指定されたコードに対応するフォントビットイメージをバッファ *font* に格納します。
引数 *font* のレイアウトは次のようになります。

hh：フォントの高さ　　ww：フォントの幅

バッファサイズの最大長は34バイトです。また，引数 *code* は次のような形式です。

引数 *code*		文字	高さ	幅	引数 *font*
上位 8	**下位 8**	**タイプ**	**hh**	**ww**	**サイズ**
0x80	ANKc	HCRT	2	1	18
0x00	ANKc	NCRT	1	1	10
shift-JISc		全角	2	2	34
0x29	0xXX	半角	2	1	18
0x2A	0xXX	半角	2	1	18
0x00	ANKc	1/4角	1	1	10

HCRT：高解像度 CRT
NCRT：標準 CRT
ANKc：8ビット JIS コード
shift-JISc：シフト JIS コード
0xXX：有効な8ビット JIS コード

この関数は，PC-9801の CRT-BIOS を使用しています。

戻り値　　ありません。

可搬性　　この関数は PC-9801シリーズでのみ動作します。

関連項目　　**putuserfont**

例

```
#define  HH     gb[0] * 8
#define  WW     gb[1]
#define  FT(x)  gb[2 + (x)]

void main()
{
   int i, w, h;
   unsigned char gb[64];

   getfont(0xEB9F, gb);           /* フォントの読み出し */
   for ( h = 0; h < HH; h++ ) {
      printf( "%02X", h);         /* 行番号の表示 */
      for ( w = 0; w < WW; w++ ) {
         for ( i = 0; i < 8; i++ ) {
            if (((FT(h*WW + w) << i) & 0x0080) == 0x0080)
               printf("●");      /* ビットオンの表示 */
            else
               printf("○");      /* ビットオフの表示 */
         }
      }
      printf( "¥n");
   }
}
```

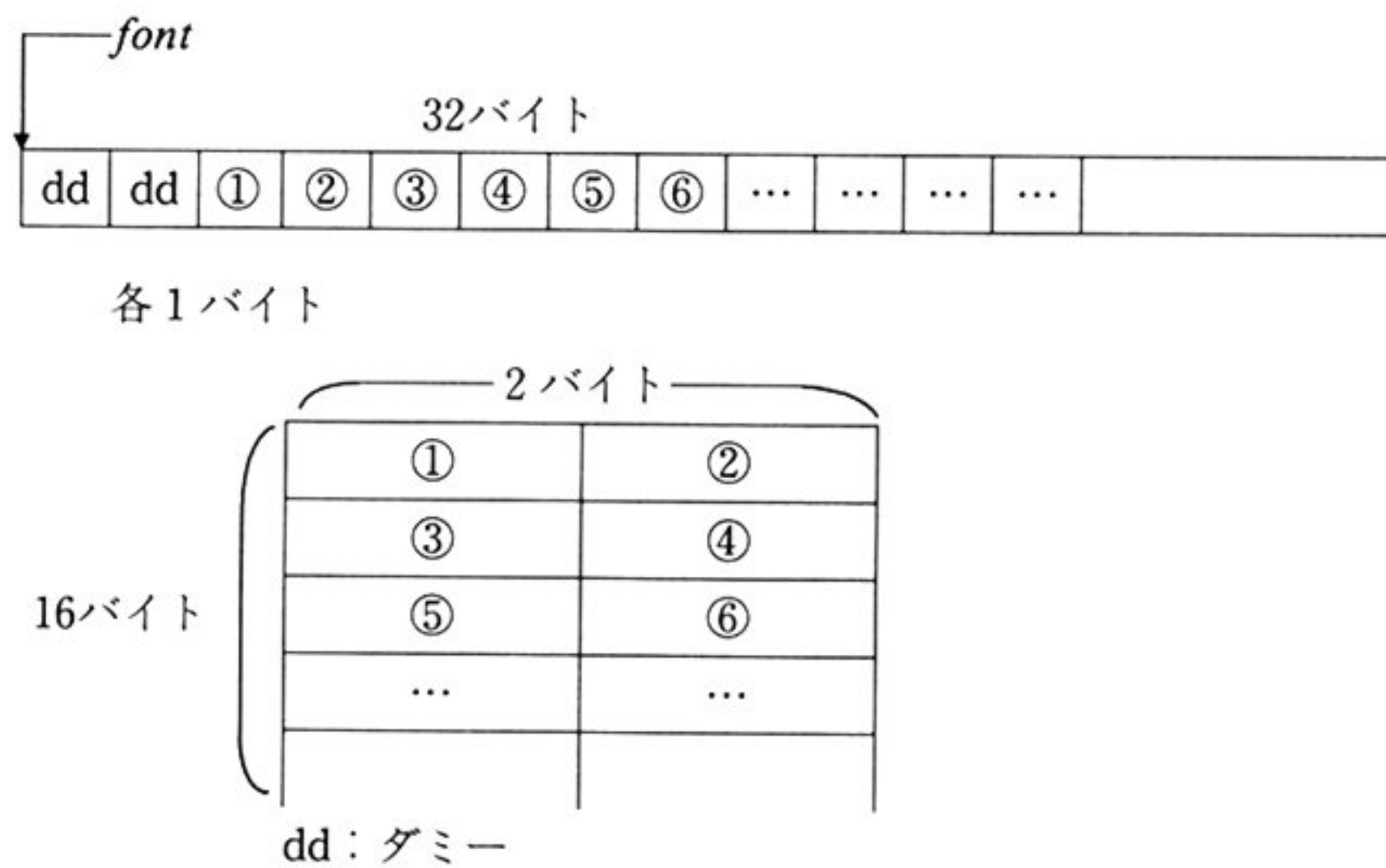
2

putuserfont

機能 ユーザフォントパターンを定義します。

形式 void putuserfont(unsigned int *code*, unsigned char * *font*);

プロトタイプ dos.h

機能説明 **putuserfont** は，ユーザが作成したフォントパターンをマシンのフォント専用 RAM に定義します。定義できる文字の大きさは全角のみです。引数 *code* で指定されたコードに対応するフォント専用 RAM に，バッファ *font* 内のフォントビットイメージを格納します。
引数 *font* のレイアウトとフォント領域の関係は次のようになります。

引数 *code* はシフト JIS コードで指定します。ユーザが定義できるコードの種類は機種ごとに異なり，次のとおりです。

マシンタイプ	JIS	シフト JIS
UV/VF/VM	0x7621～0x767E	0xEB9F～0xEBFC
	0x7721～0x777E	0xEC40～0xEC9E
E/F/U2	0x7621～0x765F	0xEB9F～0xEBDD

この関数は PC-9801の CRT-BIOS を使用しています。

戻り値 ありません。

可搬性 この関数は PC-9801シリーズでのみ動作します。ただし PC-9801の最初期型では動作しません。

関連項目 **getfont**

例

```
static unsigned char fontp[] = {
   0x00, 0x00,
   0xff, 0xff, 0xff, 0xff, 0x80, 0x80, 0x80, 0x80,
   0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
   0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
   0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0xff, 0xff
};

void main()
{
   putuserfont(0xEB9F, fontp);      /* フォントの定義 */
   /* ... */
}
```

IBM PC の ROM-BIOS インターフェース

IBM PC 用 ROM-BIOS インターフェースルーチンは，マシンの ROM チップに組み込まれているハードウェア基本制御ルーチンを駆動するためのインターフェースルーチンです。これらの関数を使用すれば，マシンの**機能**を最大限に引き出すことができます。ただし，他機種との互換性はまったくないので，可搬性のあるプログラムを作成しようとするのであれば，このインターフェースは使用すべきではありません。ここで説明するすべての関数は，IBM PC シリーズでのみ使用することができます。

bioscom

機能　シリアル I/O を操作します。

形式　int bioscom(int *cmd*, char *abyte*, int *port*);

プロトタイプ　bios.h

解説　**bioscom** は，*port* に与えられた I/O ポートを通じて，さまざまな RS-232 コミュニケーションを行ないます。
port の値0は COM1に，値1は COM2に，というように対応しています。
cmd の値は，以下のいずれかになります。

0	コミュニケーションパラメータを *abyte* の値にセットします。
1	*abyte* 中の文字を通信回線を通して送信します。
2	通信回線から1文字を受信します。
3	コミュニケーションポートの現在のステータスを返します。

abyte は，以下のビットの組み合わせです（各グループ中から1つの値が選択されます)。

0x02	データビット7	0x00	110ボー
0x03	データビット8	0x20	150ボー
		0x40	300ボー
0x00	ストップビット1	0x60	600ボー
0x04	ストップビット2	0x80	1200ボー
		0xA0	2400ボー
0x00	ノーパリティ	0xC0	4800ボー
0x08	奇数パリティ	0xE0	9600ボー
0x18	偶数パリティ		

たとえば，*abyte* に0xEB(0xE0 | 0x08 | 0x00 | 0x03)を与えると，コミュニケーションポートに，9600ボー，奇数パリティ，ストップビット1，データビット8をセットすることになります。**bioscom** は，BIOS 0x14割り込みを使用します。

戻り値

cmd のすべての値に対して，**bioscom** は16ビット整数を返します。このうち，上位8ビットはステータスで，下位8ビットは *cmd* の値によって変化します。戻り値の上位8ビットは次のように定義されています。

ビット15	タイムアウト
ビット14	トランスミットシフトレジスタが空
ビット13	トランスミットホールドレジスタが空
ビット12	ブレークを検出
ビット11	フレームエラー
ビット10	パリティエラー
ビット9	オーバーランエラー
ビット8	データレディ

abyte の値が送信できなかった場合は，ビット15がセットされます。そうでない場合には，残りのビットおよび下位8ビットは適切な値にセットされます。

cmd の値が2(受信)の場合，エラーがなければ，読み込まれたバイトは戻り値の下位ビットにあります。エラーがあった場合は，上位8ビット中の少なくとも1つがセットされます。上位8ビットがどれもセットされていなければ，そのバイトはエラーなしで受信されたことになります。

cmd の値が0または3の場合，戻り値の上位8ビットは上の定義通りにセットされ，下位8ビットは以下のように定義されます。

ビット7	受信済み回線信号を検出
ビット6	リングインジケータ
ビット5	データセットレディ
ビット4	クリアトゥーセンド
ビット3	受信回線信号中の変更検出子
ビット2	トレーリングエッジリング検出子
ビット1	データセットレディ中の変更
ビット0	クリアトゥーセンド中の変更

可搬性　**bioscom** は IBM PC とその互換機でのみ動作します。

例

```
#include <bios.h>
#include <conio.h>

#define COM1 0
#define DATA_READY 0x100
/* 1200ボー, 7ビット, 1ストップ, ノーパリティ */
#define SETTINGS (0x80|0x02|0x00|0x00)

main()
{
   int register in, out, status;

   bioscom(0, SETTINGS, COM1);
   cprintf("... BIOSCOM [ESC] to exit ...¥n");
   while (1)
   {
      status = bioscom(3, 0, COM1);
      if (status & DATA_READY)
         if ( (out = bioscom(2, 0, COM1) & 0x7F) != 0 )
            putch(out);
         if (kbhit())
         {
            if ( (in = getch()) == '¥x1B')
               return(0);
            bioscom(1, in, COM1);
         }
   }
}
```

biosdisk

機能　BIOSのディスクサービスです。

形式　int biosdisk(int *cmd*, int *drive*, int *head*, int *track*,
int *sector*, int *nsects*, void * *buffer*);

プロトタイプ　bios.h

解説　**biosdisk**は，割り込み0x13を使用してディスクの操作をBIOSに直接指示します。

*drive*は，どのドライブを使用するかを指定する数値です。0が1番目のフロッピィドライブ，1が2番目のフロッピィドライブ，2が3番目のフロッピィドライブ，・・・・・となります。ハードディスクについては，*drive*の値0x80が1番目のドライブ，0x81が2番目，0x82が3番目，・・・・・を指定することになります。

ハードディスクに対しては，ディスクのパーティションではなく，物理ドライブを指定します。必要であれば，ユーザプログラム側でパーティションテーブルを読み換えなければなりません。

*cmd*は，実行する操作を示します。*cmd*の値によって，他のパラメータが必要かそうでないかがきまります。

以下に示したのは，IBM PC，XT，AT，PS/2，あるいはそれらの互換機において有効な*cmd*の値です。

0　ディスクシステムをリセットします。ドライブコントローラを強制的にハードリセットさせます。他のパラメータはすべて無視されます。

1　前回のディスク操作のステータスを返します。他のパラメータはすべて無視されます。

2　1つ以上のディスクセクタをメモリに読み込みます。読み込む先頭のセクタは，*head*，*track*，*sector*によって与えられます。セクタ数は*nsects*で与えられます。データは，1セクタ512バイトで，*buffer*に書き込まれます。

3　1つ以上のディスクセクタにメモリから書き込みます。書き込む先頭のセクタは，*head*，*track*，*sector* によって与えられます。セクタ数は *nsects* で与えられます。データは，1セクタ512バイトで，*buffer* から書き込まれます。

4　1つ以上のセクタをベリファイします。開始セクタは，*head*，*track*，*sector* によって与えられます。セクタ数は *nsects* で与えられます。

5　1つのトラックをフォーマットします。トラックは，*head* と *track* によって与えられます。*buffer* は，指定されたトラック上に書き込まれるセクタヘッダのテーブルを指します。このテーブルおよびフォーマット操作の詳細については，IBM PC のテクニカルリファレンスマニュアルを参照してください。

以下に示したのは，IBM XT，AT，PS/2，あるいはそれらの互換機においてのみ有効な *cmd* の値です。

6　トラックをフォーマットし，バッドセクタフラグをセットします。

7　指定のトラックからドライブをフォーマットします。

8　現在のドライブパラメータを返します。ドライブ情報は，*buffer* 中の最初の4バイトに返されます。

9　ドライブペア性質を初期化します。

10　ロングリード（1セクタ512バイト＋4エキストラバイト）を行ないます。

11　ロングライト（1セクタ512バイト＋4エキストラバイト）を行ないます。

12　ディスクシークを行ないます。

13　ディスクリセットを行ないます。

14　セクタバッファを読みます。

15　セクタバッファを書きます。

16　指定のドライブがレディかどうかを調べます。

17　ドライブを再検査します。

18　コントローラ RAM を診断します。

19　ドライブを診断します。

20　コントローラ内部を診断します。

注意：**biosdisk** は，ファイルより下のレベルで，セクタをじかに操作するため，はーどディスク上のファイルの内容やディレクトリを壊してしまう可能性があります。

戻り値　　**biosdisk** は，以下のビットからなるステータスバイトを返します。

0x00	操作に成功。
0x01	不正なコマンド。
0x02	アドレスマークが見つからない。
0x03	書き込み禁止ディスクに書き込もうとした。
0x04	セクタが見つからない。
0x05	リセットに失敗（ハードディスク)。
0x06	前回操作以降にディスクが交換された。
0x07	ドライブパラメータの有効化に失敗。
0x08	DMA のオーバーラン。
0x09	64K 境界にまたがる DMA を行なおうとした。
0x0A	不良セクタが検出された。
0x0B	不良トラックが検出された。
0x0C	サポートされないトラック。
0x10	ディスク読み込み中に不正な CRC/ECC。
0x11	CRC/ECC がデータエラーを修正した。
0x20	コントローラが動作に失敗した。
0x40	シーク操作に失敗した。
0x80	アタッチメントが応答に失敗した。
0xAA	ドライブノットレディ（ハードディスクのみ)。
0xBB	未定義のエラーが発生（ハードディスクのみ)。
0xCC	ライトフォルトが発生。
0xE0	ステータスエラー。
0xFF	センス操作に失敗。

0x11は，そのデータは正しいのでエラーではないことに注意してください。いずれにせよその値は返されるので，アプリケーション側でそれが正しいかどうかを決めることができます。

可搬性　　**biosdisk** は，IBM PC とその互換機でのみ動作します。

関連項目　　absread，asbwrite

biosequip

機能　マシンの装備をチェックします。

形式　int biosequip(void);

プロトタイプ　bios.h

解説　**biosequip** は，現在システムに接続されている機器情報を返します。

注意：この関数は BIOS インターフェースではありません。

戻り値　16ビット中に下記のように情報が返されます。

ビット14-15　組み込まれているパラレルプリンタの数
ビット13　シリアルプリンタの接続
ビット12　ゲーム I/O の接続
ビット9-11　送信ポートの数
ビット10　DMA の有無
　0 = マシンは DMA を持っている。
　1 = マシンは DMA を持っていない（PC Jr など)。
ビット6-7　ディスクの数
　00 = 1ドライブ
　01 = 2ドライブ
　10 = 3ドライブ
　11 = 4ドライブ（ビット0が1のときのみ）
ビット5　イニシャル
ビット4　ビデオモード
　00 = 未使用
　01 = 40×25BW，カラーカードつき
　10 = 80×25BW，カラーカードつき
　11 = 80×25BW，モノカードつき

ビット2-3　　マザーボードのRAMサイズ

00 = 16K

01 = 32K

10 = 48K

11 = 64K

ビット1　　浮動小数点コプロセッサ

ビット0　　ディスクからのブート

可搬性　　**biosequip**は，IBM PCとその互換機でのみ動作します。

bioskey

機能　キーボードインターフェースです。

形式　int bioskey(int *cmd*);

プロトタイプ　bios.h

解説　**bioskey** は，BIOS 割り込み0x16を使って，さまざまなキーボード操作を行ないます。パラメータ *cmd* によって行なわれる操作が決まります。

戻り値　**bioskey** が返す値は，*cmd* によって決定される操作に依存します。

cmd	**bioskey で行なわれる操作**
0	下位8ビットがゼロでない場合，**bioskey** は，キュー中に待機している次のキーストロークの ASCII 文字，あるいはキーボード上で次に押されたキーを返します。下位8ビットがゼロの場合は，上位8ビットは，IBM PC のテクニカルリファレンスマニュアルで定義されている拡張キーボードコードになります。
1	キーストロークが読み出し可能かどうかをテストします。ゼロの戻り値は，キーが読み出し可能でないことを意味します。そうでない場合は，次のキーストロークの値が返されます。キーストロークそのものは，*cmd* の値が0の次の **bioskey** の呼び出しによって返すために保持されます。
2	現在のシフトキーのステータスを要求します。戻り値は，以下の値の OR をとることによって作られます。

ビット7	0x80	***Insert*** オン
ビット6	0x40	***Caps*** オン

ビット5	0x20	***Num Lock*** オン
ビット4	0x10	***Scroll Lock*** オン
ビット3	0x08	***Alt*** が押されている
ビット2	0x04	***Ctrl*** が押されている
ビット1	0x02	左 ***Shift*** が押されている
ビット0	0x01	右 ***Shift*** が押されている

可搬性

bioskey は，IBM PC とその互換機でのみ動作します。

例

```
#include <stdio.h>
#include <bios.h>
#include <ctype.h>

#define RIGHT 0x0001
#define LEFT  0x0002
#define CTRL  0x0004
#define ALT   0x0007

main()
{
   int key, modifiers;

   /* 機能1はキーが押されるまで0を返します。キーを繰り返し
      チェックすることによって入力を待ちます。              */
   while (bioskey(1) == 0) ;

   /* 機能0を使ってそのキーの戻り値を得ます。               */
   key = bioskey(0);
   printf("Key Presses was ");

   /* 機能2を使ってシフトキーが使われたかどうかを調べます  */
   modifiers = bioskey(2);
   if (mofifiers) {
      printf("[");
      if (modifiers & RIGHT) printf("RIGHT ");
      if (modifiers & LEFT ) printf("LEFT ");
      if (modifiers & CTRL ) printf("CTRL ");
      if (modifiers & ALT  ) printf("ALT ");
      printf("] ");
   }
   if (isalnum(key & 0xFF))
      printf("'%c'¥n", key);
   else
      printf("%#02x¥n", key);
}
```

biosmemory

機能　メモリサイズを返します。

形式　int biosmemory(void);

プロトタイプ　bios.h

解説　**biosmemory** は，BIOS 割り込み0x12を使用して，RAM メモリのサイズを返します。これには，ディスプレイアダプタメモリや拡張メモリは含まれません。

戻り値　**biosmemory** は，1K ブロック単位で RAM メモリのサイズを返します。

可搬性　**biosmemory** は，IBM PC とその互換機でのみ動作します。

biosprint

機能 プリンタI/Oを行ないます。

形式 int biosprint(int *cmd*, int *abyte*, int *port*);

プロトタイプ bios.h

解説 **biosprint**は，BIOS割り込み0x17を使って，*port*で指定されたプリンタに対してさまざまな操作を行ないます。
*port*の値0はLPT1に，値1はLPT2に，……というように対応しています。
cmdの値は以下のいずれかです。

0	*abyte*中の文字をプリントします。
1	プリンタポートを初期化します。
2	プリンタステータスを読み出します。

*abyte*の値は，0から255の範囲内でなければなりません。

戻り値 どの操作における戻り値も，以下のビット値のORをとって作られ，現在のプリンタステータスを示します。

ビット0	0x01	デバイスタイムアウト
ビット3	0x08	I/Oエラー
ビット4	0x10	セレクトされている
ビット5	0x20	用紙切れ
ビット6	0x40	認識
ビット7	0x80	ビジーではない

可搬性 **biosprint**は，IBM PCとその互換機でのみ動作します。

biostime

機能　BIOS タイマの読み出し/設定を行ないます。

形式　long biostime(int *cmd*, long *newtime*)；

プロトタイプ　bios.h

解説　**biostime** は，BIOS タイマの読み出しあるいは設定を行ないます。これは，深夜午前零時から1秒約18.2チックのレートで，チックを計測するタイマです。**biostime** は，BIOS 割り込み0x1A を使用します。
cmd が0の場合，**biostime** はタイマの現在の値を返します。*cmd* が1の場合には，タイマは *newtime* 中の **long** 値にセットされます。

戻り値　**biostime** が BIOS タイマを読み込んだとき（*cmd*＝0）には，タイマの現在値が返されます。

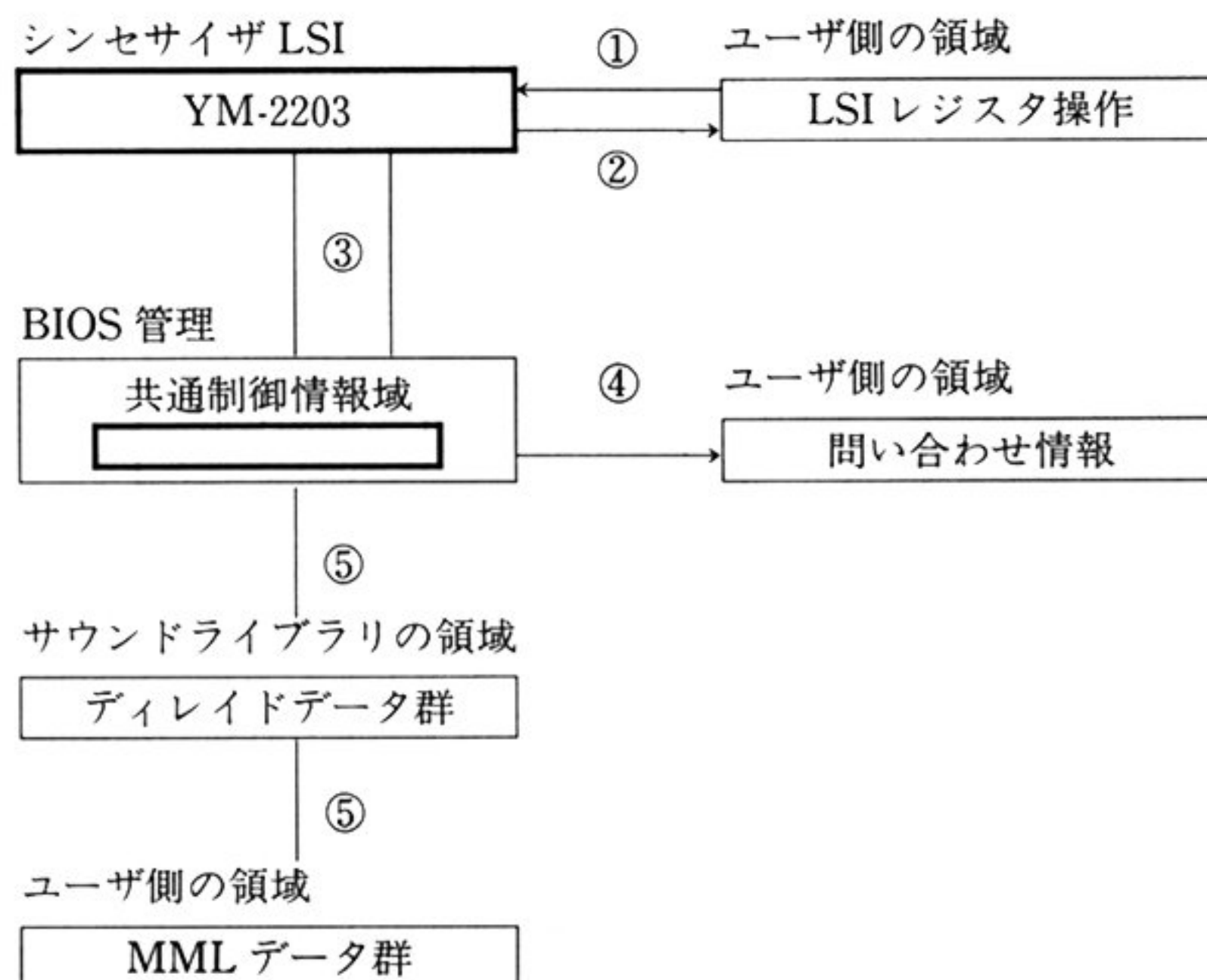
可搬性　**biostime** は，IBM PC とその互換機でのみ動作します。

PC-9801 サウンドライブラリ

Turbo C のサウンドライブラリは，NEC 製サウンドボード（PC-9801-26K）を増設したとき，または PC-9801UV2などに標準実装されているサウンド機能のためのインターフェースルーチンです。サウンドボードには YAMAHA 製のYM-2203というシンセサイザ LSI が搭載されており，FM 音源3声と SSG 音源3声の合計6声による6重和音演奏が楽しめます。

インターフェースの詳細な仕様については，サウンドボードのユーザーズマニュアル，秀和システムトレーディング株式会社の「**FM 音源スーパーサウンド**」，アスキー出版局の「**PC-9800シリーズ テクニカルデータブック**」などをご覧ください。

図 2.1 は，利用者が作成した MML フォーマットデータ（楽譜に相当する）とサウンドBIOS の関係を表わしたものです。

図 2.1　MML データとサウンド BIOS の関係

① **mc_register** の機能コード1によって，直接データを LSI のレジスタに設定します。

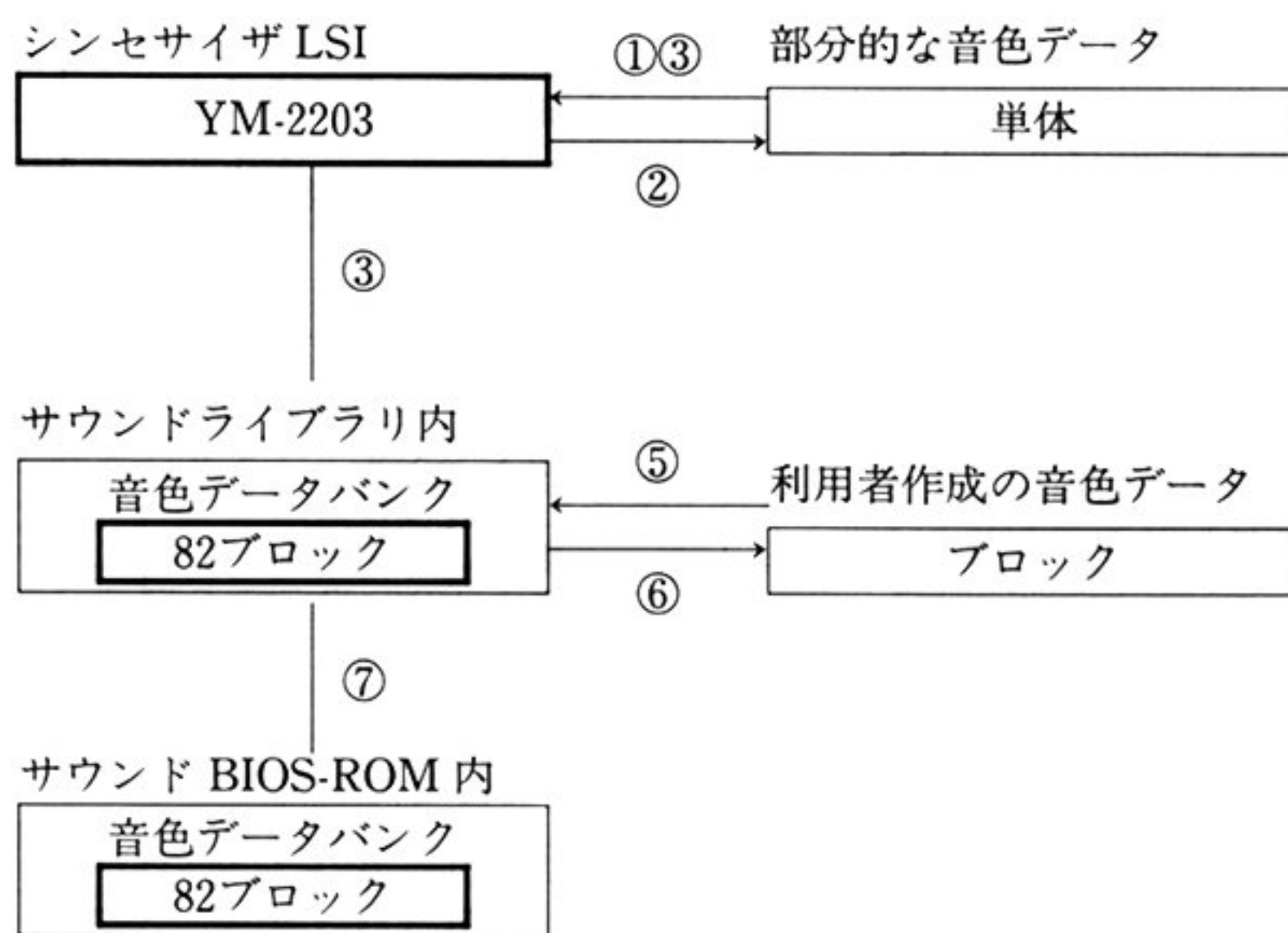
② **mc_register** の機能コード2によって，LSI のレジスタから直接データを読み込みます。

③ サウンド BIOS の割り込み制御によって，逐次，ディレイドデータ（指令）を読み出して解析し，LSI に命令を送って演奏を行ないます。また，これに伴い共通制御情報域上の管理情報を逐次更新しています。これらの動作は利用者が本来関知する必要はありません。

④ **mc_inquire** によって，共通制御情報域上の各チャンネルの制御情報を利用者側の領域に読み出します。

⑤ **mc_play** により MML フォーマットデータをディレイドフォーマットデータに変換し，サウンド BIOS に演奏を委託します。このとき，サウンド BIOS の割り込み処理がディレイドデータ（指令）を解析し，**図 2.2** の③や④のデータ転送操作を行ないます。

図 2.2 は音色データのデータ転送を行なう関数や機能の関係を表したものです。

図 2.2　音色データとデータ転送関係

① **mc_scalar** の機能コード1によって，LSI のレジスタの一部を利用者が指定した値で一時的に書き換えます。

② **mc_scalar** の機能コード2によって，LSI のレジスタの内容を利用者側の領域に読み出します。

③ MML 上に記述されたコマンド Y によって，LSI のレジスタの一部を一時的に書き換えます。これは①と同じ機能です。

④ MML 上に記述されたコマンド@x によって，ブロック単位で音色データを LSI のレジスタに設定します。

⑤ **mc_block** の機能コード1によって，利用者が作成した音色データをサウンドライブラリ内のバンクに登録します。

⑥ **mc_block** の機能コード2によって，サウンドライブラリ内の音色データバンクを利用者側の領域に読み出します。

⑦ **mc_rom** により ROM 内の音色データをサウンドライブラリ内の音色データバンクに再設定しなおします。**mc_initialize** も使用しています。

MMLをBNF（バッカス・ナウア記法）で表現したものを以下に示します。

```
<MML>       ::= [ <コマンド群> ... ] "¥0"
<コマンド群> ::=
      <音符コマンド> | <エンベロープ周期コマンド> |
      <エンベロープ形状コマンド> | <音量コマンド> |
      <省略時の音長コマンド> | <音長比率コマンド> |
      <オクターブコマンド> | <オクターブ上下コマンド> |
      <音程直接コマンド> | <テンポコマンド> | <休符コマンド> |
      <音色コマンド> | <レジスタコマンド> | <休止コマンド> |
      <移調コマンド> | <LFO効果コマンド> |
      <音色パラメータコマンド> | <演奏形態コマンド>
<音符コマンド>              ::= <音程> [ <半音階> ] [ <音長> ]
<エンベロープ周期コマンド>  ::= "M" <数値>
<エンベロープ形状コマンド>  ::= "S" <数値>
<音量コマンド>              ::= "V" <数値>
<省略時の音長コマンド>      ::= "L" <音長>
<音長比率コマンド>          ::= "Q" <数値>
<オクターブコマンド>        ::= "O" <数値>
<オクターブ上下コマンド>    ::= ">" | "<"
<音程直接コマンド>          ::= "K" <数値>
<テンポコマンド>            ::= "T" <数値>
<休符コマンド>              ::= { "R" | "P" } [ <音長> ]
<音色コマンド>              ::= "@" <数値>
<レジスタコマンド>          ::= "Y" <数値> "," <数値>
<休止コマンド>              ::= "@W" [ <音長> ]
<移調コマンド>              ::= "_" <音程>
<LFO効果コマンド>           ::= "!" | "*"
<音色パラメータコマンド>    ::= "Z" <数値> "," <数値>
<連符コマンド>              ::= "{" <コマンド群> ... "}" [<音長>]
<演奏形態コマンド>          ::= "M" { "B" | "F" }
<音程>                      ::= "C" | "D" | "E" | "F" | "G" | "A" | "B"
<半音階>                    ::= "+" | "#" | "-"
<音長>                      ::= <数値> [ "." ] [ "^" <数値> ]
<数値>                      ::= <数字> ...
<数字>                      ::= "0" | "1" | "2" | "3" | "4" |
                                "5" | "6" | "7" | "8" | "9"
```

■文字と文字の間の空白やタブは意味を持ちません。また，文字は半角文字のみで全角文字は許されません。

■表記の中で使われている記号は，次のような意味を持っています。

[a]	aを選択します。省略してもかまいません。
{a}	aを必らず選択します。
a ¦ b	aかbどちらかを必らず選択します。
a...	aを繰り返します。

注意：サウンドライブラリの一部の関数では，データのやりとりにパケット構造体を使用しています。パケット構造体についてはこの章の「**PC-9801 の ROM-BIOS インターフェース**」の節の最初に説明があるので，表の見方などはそちらを参照してください。

mc_block

機能　音色データを設定または読み出します。

形式

```
#include <music.h>
int mc_block(MC_BLOCK * blk);
```

プロトタイプ　music.h

解説　**mc_block**は，音色データ（パラメータブロック）の設定，読み出しを行ないます。

パケット内の機能コードcmmdが1のとき，音色データ番号baseで指定されたサウンドライブラリ内の音色データバンクにパケット内の個々の項目を設定します。

機能コードが2のとき，音色データ番号baseで指定されたサウンドライブラリ内の音色データバンクからパケット内の個々の項目に読み込みます。

構造体**MC_INQUIRE**は，music.hで次のように定義されています。

```
typedef struct {
    unsigned char cmmd;           /* 機能コード */
    unsigned char stus;           /* 実行後の状態 */
    unsigned char base;           /* 音色データ番号 0〜81 */
    unsigned char fb_alg;         /* フィードバック/アルゴリズム */
    unsigned char at_r[4];        /* アタック係数 */
    unsigned char opr_msk;        /* オペレータマスク */
    unsigned char dc_r[4];        /* ディケイ係数 */
    unsigned char wav_form_lfo;   /* LFO波長 */
    unsigned char ss_r[4];        /* サスティン係数 */
    unsigned char sync_dly_lfo;   /* LFOSYNC遅延時間 */
    unsigned char rl_r[4];        /* リリース係数 */
    unsigned int  speed_lfo;      /* LFO効果の速度 */
    unsigned char ss_l[4];        /* サスティンレベル */
    unsigned char p_mod_lfo;      /* LFO効果のピッチ変調深さ */
    unsigned char op_l[4];        /* 出力レベル */
    unsigned char a_mod_lfo;      /* LFO効果の振幅変調深さ */
    unsigned char keyscl[4];      /* KEYスケーリング深さ */
    unsigned char p_mos_lfo;      /* LFO効果ピッチ変調深さ(粗調整) */
    unsigned char mult[4];        /* マルチプル */
    unsigned char resv1;          /* 予約 */
    unsigned char detun[4];       /* デチューン係数 */
    unsigned char resv2;          /* 予約 */
    unsigned char a_mos_lfo[4];   /* LFO効果振幅変調深さ(粗調整) */
} MC_BLOCK;
```

音色データの詳細はサウンドボードユーザーズマニュアルと「**FM音源スーパーサウンド**」を参照してください。

戻り値　音色データ番号baseが異常なとき，0以外を返します。また，パケット内の終了状態stusが設定されます。

■パケット内のデータの授受

MC_BLOCK	01	02
cmmd	R	R
stus	W	W
base	R	R
fb_alg	R	W
at_r	R	W
opr_msk	R	W
dc_r[4]	R	W
wav_form_lfo	R	W
ss_r	R	W
sync_dly_lfo	R	W
rl_r	R	W
speed_lfo	R	W
ss_l	R	W
p_mod_lfo	R	W
op_l	R	W
a_mod_lfo	R	W
keyscl	R	W
p_mos_lfo	R	W
mult	R	W
resv1	R	W
detun	R	W
resv2	R	W
a_mos_lfo	R	W

可搬性　　この関数はPC-9801シリーズでのみ動作します。

関連項目　　**mc_scalar，mc_rom**

mc_continue

機能 演奏を再開します。

形式 void mc_continue(void);

プロトタイプ music.h

解説 **mc_continue** は，**mc_stop** で中断されていた箇所から演奏を再開します。ただし，**mc_initialize** を実行したときは再開されません。

戻り値 ありません。

可搬性 この関数は PC-9801シリーズでのみ動作します。

関連項目 **mc_stop**

mc_ground

機能　　演奏モードを切り換えます。

形式　　int mc_ground(int *mode*);

プロトタイプ　　music.h

解説　　**mc_ground**は，指定された*mode*により演奏モードをフォアグラウンドまたはバックグラウンドモードに切り換えます。引数*mode*が1のときはフォアグラウンドモードを意味し，2のときはバックグラウンドを意味します。フォアグラウンドのときは演奏が終了するまで次の処理はを行ないません。現在のモードを知りたいときは**mc_inquire**を使用してください。

戻り値　　*mode*が範囲外のときは0以外を返します。

可搬性　　この関数はPC-9801シリーズでのみ動作します。

関連項目　　**mc_inquire**

mc_initialize

機能　サウンドライブラリの実行環境を初期化します。

形式　int mc_initialize(void)；

プロトタイプ　music.h

解説　**mc_initialize** は，サウンドライブラリの実行環境を初期化します。他のサウンド関数を実行する前に，必らずこの関数を一度実行しなければなりません。また，この関数はサウンド拡張機能を次のように初期化します。各チャンネルのディレイドデータ用領域は300バイトに固定されています。

■SSG エンベロープ　：M255，S1
■SSG 音量　：V7
■音長　：L4（R4）
■音の長さの割合　：Q7
■オクターブ　：O4
■テンポ　：T120
■FM チャンネル音色　：0（デフォルト音量）
■音色バンク　：立上げ時の状態に設定
■演奏モード　：フォアグランド，楽音モード

この関数は N88-BASIC(86)のサウンド拡張命令 PLAY ALLOC に相当します。

戻り値　サウンドボードが実装されていないときには，0以外を返します。

可搬性　この関数は PC-9801シリーズでのみ動作します。

mc_inquire

機能 演奏状況を調べます。

形式 # include 〈music.h〉

int mc_inquire(MC_INQUIRE * *inq*) ;

プロトタイプ music.h

解説 **mc_inquire** は，構造体 **MC_INQUIRE** 内に全チャンネルのエンプティ割り込み条件，未演奏データ長，カレント Key-No.，デフォルト音長，カレントタッチ，演奏中フラグとテンポ数を返します。また，メンバ maxsize，minsize に全チャンネル中で最長の未演奏データ長と最短の未演奏データ長を返し，メンバ allbusy には全チャンネル中で現在演奏中かどうかを返します。1つでも演奏中のときは0以外を，全演奏が終了しているときは0を返します。メンバ ground は現在フォアグラウンド(1)かバックグラウンド(2)であるかを示し，メンバ mode は現在の演奏モードを示しています。

構造体 **MC_INQUIRE** は，music.h で次のように定義されています。

```
typedef struct {
   unsigned char cmmd;         /* 機能コード 現在未使用 */
   unsigned char stus;         /* 実行後の状態 現在未使用 */
   int           unsize[6];    /* 各チャンネルの未演奏データ長 */
   unsigned int  empty[6];     /* 各チャンネルの割り込み条件 */
   unsigned char keyno[6];     /* 各チャンネルのカレントキー番号 */
   unsigned char length[6];    /* 各チャンネルのデフォルト音長 */
   unsigned char touch[6];     /* 各チャンネルのカレントタッチ */
   unsigned char busy[6];      /* 各チャンネルの演奏中フラグ */
   unsigned char tempo;        /* テンポ数 */
   int           maxsize;      /* 最長の未演奏データ長 */
   int           minsize;      /* 最短の未演奏データ長 */
   char          ground;       /* 現在の処理モード 1, 2 */
   char          mode;         /* 現在の演奏モード 1, 2, 3 */
   unsigned char allbusy;      /* 全チャンネルの演奏状況 */
} MC_INQUIRE;
```

戻り値　　全チャンネルの演奏状況 allbusy と同じ値を，16ビット中の下位8ビットに返します。

■パケット内のデータの授受

MC_INQUIRE	
cmmd	×
stus	×
unsize	W
empty	W
keyno	W
length	W
touch	W
busy	W
tempo	W
maxsize	W
minsize	W
ground	W
mode	W
allbusy	W

可搬性　　この関数は PC-9801シリーズでのみ動作します。

関連項目　　**mc_play，mc_ground，mc_mode**

mc_mode

機能　チャンネル3に対してモードを指定します。

形式　int mc_mode(int *fm)* ;

プロトタイプ　music.h

解説　**mc_mode** は，チャンネル3（サウンドライブラリ内ではCH2）のFM音源に対するモード指定です。引数 *fm* が1のときは楽音モード，2のときは効果音モード，3のときはCSMモードを意味します。現在のモードを知りたいときは **mc_inquire** を使用してください。

戻り値　*fm* が範囲外のときは0以外を返します。

可搬性　この関数はPC-9801シリーズでのみ動作します。

関連項目　**mc_inquire**

mc_play

機能　演奏を行ないます。

形式　# include <music.h>

int mc_play(MC_MML ＊ *mml*)；

プロトタイプ　music.h

解説　**mc_play** は，構造体 **MC_MML** 内のメンバ *mml* に与えられた MML の指示にしたがって音楽を演奏します。MML のフォーマットの詳細はサウンドボードユーザーズマニュアルを参照してください。MML を指定しないチャンネルは，NULL ポインタを設定するか，または NULL 文字列を与えます。構造体 **MC_MML** はヘッダファイル music.h に下記のように定義されています。

```
typedef struct {
   unsigned char  cmmd;        /* 機能コード 現在未使用 */
   unsigned char  stus;        /* 実行後の状態 現在未使用 */
   char          *mml[6];      /* MML文字列配列 */
   int            err_ch;      /* エラーチャンネル番号 */
   int            err_pos;     /* エラー位置 */
} MC_MML;
```

注意：チャンネル番号は通常 CH1～CH6で表現されますがサウンドライブラリ中では CH0～CH5で表現します。

この関数は N88-BASIC(86)のサウンド拡張命令 PLAY に相当します。

戻り値　正常終了したときは0を返し，指定されたMMLにてエラーが発生したときは0以外を返します。また，パケット内の変数err_chにエラーを起こしたチャンネル番号，変数err_posにエラーを起こしたMML文字列中の位置を返します。

■パケット内のデータの授受

MC_MML	
cmmd	×
stus	×
mml	R
err_ch	W
err_pos	W

可搬性　この関数はPC-9801シリーズでのみ動作します。

mc_register

機能　サウンドLSI内のレジスタを操作します。

形式　# include <music.h>

void mc_register(MC_REGISTER * *reg*);

プロトタイプ　music.h

解説　**mc_register**は，サウンドLSIの各レジスタにデータを直接設定したり，読み出しを行なったりします。

パケット内の機能コード *cmmd* が1のとき，レジスタ番号 *reg* で指定されたレジスタに設定値 *value* を設定します。

機能コード *cmmd* が2のとき，レジスタ番号 *reg* で指定されたレジスタを読み込み値 *value* に読み込みます。

構造体 **MC_REGISTER** は，music.hで次のように定義されています。

```
typedef struct {
   unsigned char cmmd;        /* 機能コード */
   unsigned char stus;        /* 実行後の状態 */
   unsigned char reg;         /* レジスタ番号 0x00～0xFF */
   unsigned char value;       /* 設定値／読込値 */
} MC_REGISTER;
```

SSG周波数，SSGノイズ，固定量/可変音量，エンベロープ周期，エンベロープ形状等の機能の操作は，この関数によって行ないます。

この関数はN88-BASIC(86)のサウンド拡張命令VOICE REGに相当します。

戻り値　　ありません。

■パケット内のデータの授受

MC_REGISTER	01	02
cmmd	R	R
stus	×	×
reg	R	R
value	R	W

可搬性　　この関数はPC-9801シリーズでのみ動作します。

mc_rom

機能　音色データバンクを再設定します。

形式　int mc_rom(int *base*, int *n*);

プロトタイプ　music.h

解説　**mc_rom** は，引数 *base* で指されるサウンド BIOS-ROM 内の音色データバンクから引数 *n* 個分を，該当するサウンドライブラリ内の音色データバンクに再設定します。

戻り値　サウンド BIOS-ROM が存在しないときは0以外を返します。*base* と *n* の関係が不正なときは0以外を返します。

可搬性　この関数は PC-9801シリーズでのみ動作します。

関連項目　**mc_block**

mc_scalar

機能　チャンネルの各パラメータのデータの設定/読み出しを行ないます。

形式　# include <music.h>

int mc_scalar(MC_SCALAR * *sca*);

プロトタイプ　music.h

解説　**mc_scalar** は，サウンド LSI の各チャンネルの各パラメータにデータを単体で設定したり，読み出しを行なったりします。

パケット内の機能コード cmmd が1のとき，チャンネル番号 ch とパラメータ番号 param で指定された機能パラメータに設定値 value を設定します。

機能コード cmmd が2のとき，チャンネル番号 ch とパラメータ番号 param で指定された機能パラメータを読み込み値 value に読み込みます。

構造体 **MC_SCALAR** は，music.h で次のように定義されています。

```
typedef struct {
    unsigned char cmmd;      /* 機能コード */
    unsigned char stus;      /* 実行後の状態 */
    unsigned char ch;        /* チャンネル番号 CH0～CH5 */
    unsigned char param;     /* パラメータ番号 0～49 */
    unsigned int  value;     /* 設定値／読込値 */
} MC_SCALAR;
```

戻り値　チャンネル番号 ch とパラメータ番号 param が異常なとき，0以外を返します。また，パケット内の終了状態 stus が設定されます。

■パケット内のデータの授受

MC_SCALAR	01	02
cmmd	R	R
stus	W	W
ch	R	R
param	R	R
value	R	W

可搬性　　この関数はPC-9801シリーズでのみ動作します。

関連項目　　**mc_block**

mc_stop

機能　演奏を中断します。

形式　void mc_stop(void);

プロトタイプ　music.h

解説　**mc_stop** は，全チャンネルの演奏を中断します。演奏中にユーザプログラムが終了するときは，通常この関数を呼び出しますが，実行しなくても演奏は自動的に中止されます。これは演奏中に ***CTRL-C*** を押した場合でも有効です。逆に演奏したままで利用者のプログラムを常駐終了させたいときは，次に示す変数をプログラムで外部定義してください。

```
int  mc_nonstop = 1;
```

通常，ユーザプログラムで最後まで演奏するときには，**mc_inquire** によって演奏が全チャンネルが終了したかチェックしてから **mc_stop** を実行します。

注意：プログラムを強制終了させるには，***CTRL-C*** を押す方法と ***STOP*** キーを押す方法があります。システムがハードディスクを備えている場合に ***STOP*** キーで強制終了させると，システムが暴走することがあるので注意してください。

戻り値　ありません。

可搬性　この関数は PC-9801シリーズでのみ動作します。

関連項目　**mc_continue**

日本語処理ライブラリ

Turbo Cは，日本語を正しく処理することができるようになっています。もう少し具体的に言えば，文字列内に含まれる全角文字（2バイトコード）を，全角文字として正しく認識し，処理するということです。また，全角文字とANK文字の判別や，全角文字として正しいコードかどうかの判定，全角文字から半角文字へ（あるいはその逆）の変換などを行なうライブラリルーチンも用意されています。

この章では，プログラムで漢字を使用する際の注意点（ほんの少しですが）を最初に述べ，そのあと日本語処理ライブラリ関数のリファレンスをのせてあります。

漢字オプション

全角文字の第1バイトは，0x81～0x9F，および0xE0～0xFCの範囲になります。この範囲のコードは，モードによってはグラフィック文字として使用されます。

全角文字の第2バイトの範囲は，0x40～0x7E，および0x80～0xFCです。この範囲のコードは，ANKの英字と記号の一部，半角のカタカナ，およびグラフィック文字と重なっています。

Cで全角文字を扱う際に問題になるのは，この第2バイト目が'¥x5C'(円記号)の場合です。円記号は，Cの文字列では通常エスケープシーケンスとして使用されているので，そのままでは文字化けなどの思わぬ結果になってしまいます。

Turbo Cには，文字列内の円記号のコードをエスケープシーケンスとしてではなく，そのまま文字列の一部として扱い，また全角文字は2バイト単位で正しく扱うようにするためのオプション（漢字オプション）が用意されています。デフォルトでは漢字オプションはオンになっており，文字列の中の0x81～0x9Fおよび0xE0～0xFCのコードは全角文字の第1バイトであると認識され，その次の1バイトは全角文字の第2バイトとみなされます。

漢字オプションをオフにすると，その範囲のコードは1バイト単位でグラフィック文字として扱われることになります。

漢字オプションは，TC(統合環境版)では，**O**ptions/**C**ompilerメニューの，**C**ode generation/**K**anji stringsです。TCC（コマンドライン版）では-Jになっています。グラフィック文字を使用する場合には，メニューオプションをオフにするか，TCCであればコマンドライン（またはTURBOC.CFGの中）で-J-を指定してください。

btom

機能　文字列の先頭からnbyteまでに存在する文字数を調べます。

形式　unsigned int btom(char * *string*, unsigned int *n*)；

プロトタイプ　jstring.h

解説　**btom**は，引数*string*で指定された文字列の先頭から*n*バイトまでに存在する文字数を調べます。全角文字も1文字として数えます。
指定した*n*バイト内にヌル文字が現われた場合は，そのヌル文字の直前までの文字数を返します。さらにこのヌル文字が漢字2バイト目にあたる場合は，その前の漢字第1バイト目もヌル文字とみなして文字数には含みません。また*n*バイト目が漢字1バイトである場合もその直前までの文字数を返します。

戻り値　**btom**は，指定された範囲の文字数を返します。*n*に0を指定すると0が返されます。

可搬性　MS-DOSに特有の関数です。

関連項目　**mtob**

chkctype

機能　文字タイプを調べます。

形式　# include <jctype.h>

int chktype(char *c*, int *type*) ;

プロトタイプ　jctype.h

解説　**chkctype** は，文字 *c* のタイプを調べます。引数 *type* によって，タイプを判定する基準を指定します。

type の値と戻り値の関係は，次のようになります。

type	戻り値	(値)	意味
1以外	CT_ANK	(0)	ANK 文字
1以外	CT_KJ1	(1)	漢字第1バイト
1	CT_KJ2	(2)	漢字第2バイト
1	CT_ILGL	(-1)	無効なタイプ

無効なタイプとは，漢字の第2バイトが **iskanji2** でないか，*c* がヌル文字である場合を意味します。

可搬性　MS-DOS に特有の関数です。

関連項目　**nthctype**

hantozen

機能　半角文字を全角文字に変換します。

形式　unsigned short hantozen(unsigned short *c*);

プロトタイプ　jctype.h

解説　**hantozen** は，半角文字 *c* を全角に変換します。ただし変換が行なわれるのは，半角文字が0x20～0x7e の範囲に限られます。

戻り値　**hantozen** は，変換が正しく行なわれれば，変換後の文字コードを返します。変換できなかった場合には，*c* をそのまま返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**zentohan**

isalkana

機能　英文字またはカナ文字かどうかを判定します。

形式

```
# include <jctype.h>
int isalkana(int c);
```

プロトタイプ　jctype.h

解説　**isalkana** マクロは，*c* が英文字またはカナ文字であるかどうかを判定します。(**isalpha**＋**iskmoji**)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有のマクロです。

isalnmkana

機能　英数字またはカナ文字かどうかを判定します。

形式

```
# include <jctype.h>
int isalnmkana(int c);
```

プロトタイプ　jctype.h

解説　**isalnmkana** マクロは，*c* が英数字またはカナ文字であるかどうかを判定します。(**isalnum**＋**iskmoji**)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有のマクロです。

isgrkana

機能 スペース以外の半角文字かどうかを判定します。

形式 # include <jctype.h>
int isgrkana(int *c)*;

プロトタイプ jctype.h

解説 **isgrkana** マクロは，*c* がスペース以外の半角文字かどうかを判定します。
(**isgraph**＋**iskana**)

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有のマクロです。

iskana

機能 半角のカナコードかどうかを判定します。

形式 # include <jctype.h>
int iskana(int *c)*;

プロトタイプ jctype.h

解説 **iskana** マクロは，*c* が半角のカナコードかどうかを判定します。
($0xA1 \leqq c \leqq 0xDF$)

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有のマクロです。

iskanji

機能 漢字第1バイトかどうかを判定します。

形式

```
#include <jctype.h>
int iskanji(int c);
```

プロトタイプ jctype.h

解説 **iskanji** マクロは，*c* が漢字第1バイトかどうかを判定します。
（0x81 ≦ *c* ≦ 0x9F　または　0xE0 ≦ *c* ≦ 0xFC）

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有のマクロです。

iskanji2

機能 漢字第2バイトかどうかを判定します。

形式

```
#include <jctype.h>
int iskanji2(int c);
```

プロトタイプ jctype.h

解説 **iskanji2** マクロは，*c* が漢字第2バイトかどうかを判定します。
（0x40 ≦ *c* ≦ 0x7E　または　0x80 ≦ *c* ≦ 0xFC）

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有のマクロです。

iskmoji

機能　半角のカナ文字かどうかを判定します。

形式　# include <jctype.h>
int iskmoji(int *c)*;

プロトタイプ　jctype.h

解説　**iskmoji** マクロは，*c* が半角カナ文字かどうかを判定します。
(0xA6 ≦ *c* ≦ 0xDF)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有のマクロです。

iskpun

機能　半角のカナ句読点かどうか判定します。

形式　# include <jctype.h>
int iskpun(int c)；

プロトタイプ　jctype.h

解説　**iskpun** マクロは，*c* がカナ句読点かどうかを判定します。
(0xA1 ≦ *c* ≦ 0xA5)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有のマクロです。

ispnkana

機能　英句読点またはカナ句読点であるかどうかを判定します。

形式　# include <jctype.h>
int ispnkana(int *c)*；

プロトタイプ　jctype.h

解説　**ispnkana** マクロは，*c* が英句読点またはカナ句読点であるかどうかを判定します。(**ispunct** ＋ **iskpun**)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有のマクロです。

isprkana

機能　半角文字かどうかを判定します。

形式　# include <jctype.h>
int isprkana(int *c)*；

プロトタイプ　jctype.h

解説　**isprkana** マクロは，*c* が半角文字かどうかを判定します。
(**isprint** ＋ **iskana**)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有のマクロです。

jasctime

機能　**asctime** の日本語対応版です。

形式

```
#include <time.h>
char *jasctime(const struct tm *tm);
```

プロトタイプ　time.h

解説　**jasctime** は，**asctime** の日本語対応版で，変換後の文字列は次の形式になります。

```
1988年09月23日 (金) 07:15:06¥n¥0
```

詳細については **asctime** を参照してください。

戻り値　変換後の文字列へのポインタを返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**asctime, ctime, jctime**

jctime

機能 **ctime** の日本語対応版です。

形式

```
#include <time.h>
char * jctime(const time_t * clock);
```

プロトタイプ time.h

解説 **jctime** は，**ctime** の日本語対応版で，変換後の文字列は次の形式になります。

```
1988年09月23日 (金) 07:15:06¥n¥0
```

詳細については **ctime** を参照してください。

戻り値 変換後の文字列へのポインタを返します。

可搬性 MS-DOS に特有の関数です。

関連項目 **asctime**, **ctime**, **jasctime**

jisalpha

機能　全角英文字かどうかを判定します。

形式　int jisalpha(unsigned short *c*);

プロトタイプ　jctype.h

解説　**jisalpha** は，*c* が全角英文字かどうかを判定します。
(0x8260 ≦ *c* ≦ 0x8279　または　0x8281 ≦ *c* ≦ 0x829A)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有の関数です。

jisdigit

機能　全角数字かどうかを判定します。

形式　int jisdigit(unsigned short *c*);

プロトタイプ　jctype.h

解説　**jisdigit** は，*c* が全角数字かどうかを判定します。
(0x824F ≦ *c* ≦ 0x8258)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有の関数です。

jishira

機能 全角ひらがなかどうかを判定します。

形式 int jishira(unsigned short *c*)；

プロトタイプ jctype.h

解説 **jishira** は，*c* が全角ひらがなかどうかを判定します。
(0x829F ≦ c ≦ 0x82F1)

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有の関数です。

関連項目 **jiskana, jiskata**

jiskana

機能 全角カタカナかどうかを判定します。

形式 int jiskana(unsigned short *c*);

プロトタイプ jctype.h

解説 **jiskana**は，*c*が全角カタカナかどうかを判定します。
(0x8340 ≦ *c* ≦ 8396)

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOSに特有の関数です。

関連項目 **jishira, jiskata**

jiskata

機能　全角カタカナかどうかを判定します。

形式　int jiskata(unsigned short *c*)；

プロトタイプ　jctype.h

解説　**jiskata** は，*c* が全角カタカナかどうかを判定します。
(0x8340 ≦ *c* ≦ 8396)
この関数は，**jiskana** とまったく同じです。

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**jishira**, **jiskana**

jiskigou

機能 全角句読点かどうかを判定します。

形式 int jiskigou(unsigned short *c*);

プロトタイプ jctype.h

解説 **jiskigou** は，*c* が全角句読点かどうかを判定します。
(0x8141 ≦ *c* ≦ 0x81AC)

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有の関数です。

jisl0

機能 シフト JIS コード中の漢字以外の文字かどうかを判定します。

形式 int jisl0(unsigned short *c*);

プロトタイプ jctype.h

解説 **jisl0** は，*c* がシフト JIS コードの中の漢字以外かどうかを判定します。
(0x8140 ≦ *c* ≦ 0x889E)

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有の関数です。

関連項目 **jisl1, jisl2**

jisl1

機能　JIS 第1水準漢字かどうかを判定します。

形式　int jisl1(unsigned short *c*)；

プロトタイプ　jctype.h

解説　**jisl1** は，*c* が JIS 第1水準の漢字かどうかを判定します。
（0x889F ≦ *c* ≦ 0x9872）

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**jisl0, jisl2**

jisl2

機能　JIS 第2水準漢字かどうかを判定します。

形式　int jisl2(unsigned short *c*)；

プロトタイプ　jctype.h

解説　**jisl2** は，*c* が JIS 第2水準の漢字かどうかを判定します。
(0x989F ≦ *c* ≦ 0xEA9E)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**jisl0, jisl1**

jislower

機能　全角英小文字かどうかを判定します。

形式　int jislower(unsigned short *c*)；

プロトタイプ　jctype.h

解説　**jislower** は，*c* が全角英小文字かどうかを判定します。
(0x8281 ≦ *c* ≦ 0x829A)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**jisupper**

jisspace

機能　全角スペースかどうかを判定します。

形式　int jisspace(unsigned short *c*)；

プロトタイプ　jctype.h

解説　**jisspace** は，*c* が全角スペースかどうかを判定します。
(*c* ＝ 0x8140)

戻り値　条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性　MS-DOS に特有の関数です。

jistojms

機能 JIS コードをシフト JIS コードに変換します。

形式 unsigned short int jistojms(unsigned short int *ch)*;

プロトタイプ jctype.h

解説 **jistojms** は，JIS コード *ch* をシフト JIS コードに変換します。
ch の上位バイトに JIS コードの第1バイト，*ch* の下位バイトに JIS コードの第2バイトが入っていなければなりません。

戻り値 *ch* が JIS コードとして適切な値であれば，対応するシフト JIS コードを返します。*ch* が適切でない場合は0を返します。

可搬性 MS-DOS に特有の関数です。

関連項目 **jmstojis**

jisupper

機能 全角英大文字かどうかを判定します。

形式 int jisupper(unsigned short *c*);

プロトタイプ jctype.h

解説 **jisupper** は，*c* が全角英大文字かどうかを判定します。
(0x8260 ≦ *c* ≦ 0x8279)

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有の関数です。

関連項目 **jislower**

jiszen

機能 全角文字かどうかを判定します。

形式 int jiszen(unsigned short *c*);

プロトタイプ jctype.h

解説 **jiszen** は，*c* が全角文字かどうかを判定します。
(0x8140 ≦ *c* ≦ 0xEFFC)

戻り値 条件を満たせば0以外の値を，そうでなければ0を返します。

可搬性 MS-DOS に特有の関数です。

jmstojis

機能　シフト JIS コードを JIS コードに変換します。

形式　unsigned short int jmstojis(unsigned short int *ch*)；

プロトタイプ　jctype.h

解説　**jmstojis** は，シフト JIS コード *ch* を JIS コードに変換します。
ch の上位バイトにシフト JIS コードの第1バイト，*ch* の下位バイトにシフト JIS コードの第2バイトが入っていなければなりません。

戻り値　*ch* がシフト JIS コードとして適切な値であれば，対応する JIS コードを返します。*ch* が適切でない場合は0を返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**jmstojis**

jstradv

機能　文字列ポインタを移動させます。

形式　char ＊ jstradv(unsigned char ＊ *s*, unsigned int *n*)；

プロトタイプ　jstring.h

解説　**jstradv** は，*s* が指す文字列のポインタを n 文字だけ移動し，そのポインタを返します。全角文字も1文字として数えます。
指定した文字数内にヌル文字が現われた場合は，そのヌル文字を示すポインタを返します。さらにヌル文字が漢字2バイト目にあたる場合は，その前の漢字第1バイト目もヌル文字とみなし，第1バイト目を示すポインタを返します。

戻り値　*n* が1以上であればポインタを返し，0の場合 *s* のポインタを返します。

可搬性　MS-DOS に特有の関数です。

jstrchr

機能　文字列中の指定文字の位置を調べます。

形式　char * jstrchr(char * *s*, unsigned short *c*)；

プロトタイプ　jstring.h

解説　**jstrchr** は，**strchr** の漢字対応版で，文字列 *c* で指定された文字を調べ，*c* が現われた位置のポインタを返します。
c に全角文字を指定するときは，上位8ビットに漢字第1バイト，下位8ビットに第2バイトが入ります。半角文字のときは，下位8ビットに値が入り，上位8ビットは0が入ります。

戻り値　文字列中で現われた c のポインタを返し，見つからない場合は NULL ポインタを返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**strchr**

jstrcmp

機能　文字列を比較します。

形式　int jstrcmp(char ＊ *s1*, char ＊ *s2*)

プロトタイプ　jstring.h

解説　**jstrcmp** は，**strcmp** の漢字対応版で，全角文字も1文字として数えます。文字列を比較する際文字の大小関係は，次に指定する順番になります。

　ANK ＜ カナ ＜ 漢字

戻り値　**strcmp** を参照してください。

可搬性　MS-DOS に特有の関数です。

関連項目　**strcmp**

jstrlen

機能　　文字列の文字数を調べます。

形式　　unsigned int jstrlen(char * *s*)；

プロトタイプ　　jstring.h

解説　　**jstrlen** は，引数 *s* が指す文字列の文字数を返します。漢字も1文字として数えます。ヌル文字は，文字数に含まれません。また，ヌル文字が漢字2バイト目にあたる場合はその前の漢字第1バイト目もヌル文字とみなし，文字数には含めません。

戻り値　　*s* の文字数を返します。

可搬性　　MS-DOS に特有の関数です。

関連項目　　**strlen**

jstrmatch

機能 **strpbrk** の漢字対応版です。

形式 char * jstrmatch(char * *s1*, char * *s2*)；

プロトタイプ jstring.h

解説 **jstrmatch** は，**strpbrk** の漢字対応版です。引数 *s1*，*s2* ともに漢字を使用することができます。

戻り値 *s1* の中で，*s2* の文字列内に含まれる文字のどれかが最初に現われた位置を指すポインタを返します。一致する文字がない場合には，NULL ポインタを返します。

可搬性 MS-DOS に特有の関数です。

関連項目 **strpbrk**

jstrncat

機能 **strncat** の漢字対応版です。

形式 char * jstrncat(char * *s1*, char * *s2*, unsigned int *n*) ;

プロトタイプ jstring.h

解説 **jstrncat** は，**strncat** の漢字対応版で，漢字を1文字として数えることを除いて，まったく同じ処理を行ないます。漢字の第2バイト目がヌル文字であれば，その前の第1バイト目もヌル文字とみなされます。

戻り値 **strncat** を参照してください。

可搬性 MS-DOS に特有の関数です。

関連項目 **strncat**

jstrncmp

機能　**strncmp** の漢字対応版です。

形式　char ＊ jstrncmp(char ＊ *s1*, char ＊ *s2*, unsigned int *n*)；

プロトタイプ　jstring.h

解説　**jstrncmp** は，**strncmp** の漢字対応版で，漢字を1文字として数える他は同じ処理をします。漢字の第2バイト目がヌル文字であれば，その前の第1バイト目もヌル文字とみなされます。文字列を比較する際，文字の大小関係は以下に指定する順番になります。

ANK 文字 < カナ文字 < 漢字

戻り値　**strncmp** を参照してください。

可搬性　MS-DOS に特有の関数です。

関連項目　**strncmp**

jstrncpy

機能　**strncpy** の漢字対応版です。

形式　char ＊ jstrncpy(char ＊ *s1*, char ＊ *s2*, unsigned int *n*)；

プロトタイプ　jstring.h

解説　**jstrncpy** は，**strncpy** の漢字対応版で，漢字を1文字として数える他は同じ処理をします。漢字の第2バイト目がヌル文字であれば，その前の第1バイト目もヌル文字とみなされます。

戻り値　**strncpy** を参照してください。

可搬性　MS-DOS に特有の関数です。

関連項目　**strncpy**

jstrrchr

機能　**strrchr** の漢字対応版です。

形式　char ＊ jstrrchr(char ＊ *s*, unsigned int *c*)；

プロトタイプ　jstring.h

解説　**jstrrchr** は，**strrchr** の漢字対応版で，文字列 *s* で指定された文字列を調べて，*c* が最後に現われた位置を指すポインタを返します。*c* に全角文字を指定するときは，下位8ビットに漢字第1バイト，下位8ビットに第2バイトが入ります。半角文字のときは，下位8ビットに値が入り，上位8ビットは0が入ります。

戻り値　文字列中で最後に現われた *c* へのポインタを返し，見つからない場合はNULL ポインタを返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**strrchr**

jstrrev

機能　**strrev** の漢字対応版です。

形式　char ＊ jstrrchr(char ＊ *s*)；

プロトタイプ　jstring.h

解説　**jstrrchr** は，**strrev** の漢字対応版で，文字列 *s* に全角文字が使えます。

戻り値　**strrev** を参照してください。

可搬性　MS-DOS に特有の関数です。

関連項目　**strrev**

jstrskip

機能　　指定文字列内に検索文字列に含まれない文字があるか調べます。

形式　　char * jstrskip(char * *s1*, char * *s2*)

プロトタイプ　jstring.h

形式　　**jstrskip** は，文字列 *s1* 内に，文字列 *s2* 中に含まれていない文字があった場合，その文字を指すポインタを返します。引数 *s1*，*s2* ともに全角文字を使用できます。*s1* の文字列内にヌル文字があれば，そのヌル文字を指すポインタを返します。

戻り値　　*s1* の中で，*s2* に含まれない文字が現われた位置を示すポインタを返します。

可搬性　　MS-DOS に特有の関数です。

jstrstr

機能 **strstr** の漢字対応版です。

形式 char * jstrstr(char * *s1*, char * *s2*)

プロトタイプ jstring.h

形式 **jstrstr** は，**strstr** の漢字対応版で，引数 *s1* と *s2* に全角文字が使えます。

戻り値 **strstr** を参照してください。

可搬性 MS-DOS に特有の関数です。

関連項目 **strstr**

jstrtok

機能 **strtok** の漢字対応版です。

形式 char * jstrtok(char * *s1*, char * *s2*)；

プロトタイプ jstring.h

解説 **jstrtok** は，**strtok** の漢字対応版で引数 *s1*，*s2* に全角文字が使えます。

戻り値 **strtok** を参照してください。

可搬性 MS-DOS に特有の関数です。

関連項目 **strtok**

jtohira

機能　全角カタカナを全角ひらがなに変換します。

形式　unsigned short jtohira(unsigned short *c*)；

プロトタイプ　jctype.h

解説　**jtohira** は，全角カタカナを全角ひらがなに変換します。
c には全角文字コードを指定することができます。

戻り値　変換が正しく行なわれれば変換された文字コードを返します。変換できないときには，*c* の値をそのまま返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**jtokana**, **jtokata**

jtokana

機能　全角ひらがなを全角カタカナに変換します。

形式　unsigned short jtokana(unsigned short *c*)；

プロトタイプ　jctype.h

解説　**jtokana** は，全角ひらがなを全角カタカナに変換します。
c には全角文字コードを指定することができます。

戻り値　変換が正しく行なわれれば，変換された文字コードを返します。変換できないときには，*c* の値をそのまま返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**jtohira, jtokata**

jtokata

機能　全角ひらがなを全角カタカナに変換します。

形式　unsigned short jtokata(unsigned short *c*);

プロトタイプ　jctype.h

解説　**jtokana** は，全角ひらがなを全角カタカナに変換します。
c には全角文字コードを指定することができます。
この関数は，**jtokana** とまったく同じです。

戻り値　変換が正しく行なわれれば，変換された文字コードを返します。変換できないときには，*c* の値をそのまま返します。

可搬性　MS-DOS に特有の関数です。

関連項目　**jtohira**, **jtokana**

jtolower

機能　全角英大文字を全角英小文字に変換します。

形式　unsigned short jtolower(unsigned short *c*);

プロトタイプ　jctype.h

解説　**jtolower** は，全角英大文字を全角英小文字に変換します。
c には全角文字コードを与えます。

戻り値　変換が正しく行なわれれば，変換された文字コードを返します。変換できないときには *c* の値がそのまま返ります。

可搬性　MS-DOS に特有の関数です。

関連項目　**jtoupper**, **tolower**

jtoupper

機能 全角英小文字を全角英大文字に変換します。

形式 unsigned short jtoupper(unsigned short *c*);

プロトタイプ jctype.h

解説 **jtoupper** は，全角英小文字を全角英大文字に変換します。
c には全角文字コードを与えます。

戻り値 変換が正しく行なわれれば，変換された文字コードを返します。変換できないときには *c* の値がそのまま返ります。

可搬性 MS-DOS に特有の関数です。

関連項目 **jtolower**, **toupper**

mtob

機能 文字列の先頭から *nmoji* までに存在するバイト数を調べます。

形式 unsigned int mtob(char ＊ *string*, int *nmoji*)；

プロトタイプ jstring.h

解説 **mtob** は，引数 *string* が指す文字列の先頭から *nmoji* 個の文字が何バイトを占めるかを調べます。全角文字は2バイトとして数えます。
指定した文字数以内にヌル文字が現われた場合は，そのヌル文字の直前までのバイト数を返します。さらにヌル文字が漢字の第2バイト目にあたる場合は，その直前の漢字第1バイト目もヌル文字とみなしてバイト数には含めません。

戻り値 *nmoji* が1以上であればバイト数を返し，0の場合は0を返します。

可搬性 MS-DOS に特有の関数です。

関連項目 **btom**

nthctype

機能　文字列内の ***nbyte*** の位置にある文字タイプを調べます。

形式　
```
#include <jctype.h>
unsigned int nthctype(char * string, unsigned int nbyte);
```

プロトタイプ　jctype.h

解説　**nthctype** は，引数 ***string*** 文字列内の ***nbyte*** （***n*** バイト目）の位置にある文字タイプを調べます。文字列の先頭を指定する際は，***nbyte*** は0になります。

戻り値　文字タイプを判定して文字の属性により以下の値を返します。

戻り値	値	意味
CT _ANK	0	ANK文字
CT _KJ1	1	漢字の第1バイト
CT _KJ2	2	漢字の第2バイト
CT _ILGL	-1	無効なタイプ

無効なタイプとは，漢字の第2バイトが **iskanji2** でない場合を意味します。

可搬性　MS-DOSに特有の関数です。

zentohan

機能 全角文字を半角文字に変換します。

形式 unsigned short zentohan(unsigend short *c*)；

プロトタイプ jctype.h

解説 **zentohan** は，全角文字 *c* を半角に変換します。変換が行なわれるのは，半角文字が0x20～0x7e の範囲に限られます。

戻り値 **zentohan** は，変換が正しく行なわれれば，変換後の文字コードを返します。変換できなかった場合には，*c* をそのまま返します。

可搬性 MS-DOS に特有の関数です。

関連項目 **hantozen**

付録 A

コンパイラエラーメッセージ

Turbo Cのコンパイル診断メッセージは，致命的なエラー（Fatal error），エラー（Error），警告（Warning）の3つのクラスに分類されます。

致命的なエラーが出ることはまれで，コンパイラ内部でのエラーを示す場合がほとんどです。致命的なエラーが検出されると，コンパイルはただちに中止されます。このような場合は，適当な処置をしてから，もう一度コンパイルを行なってください。

エラーは，プログラムの構文の誤り，ディスクエラーやメモリアクセスエラー，コマンドラインの誤りを指摘するものです。コンパイラは現在行なっているフェーズが終わった段階で停止します。コンパイルの各フェーズ（前処理，解析，最適化，コード生成）では，可能な限り多くのエラーが検出されます。

警告が検出されても，コンパイルは中止されません。問題はあるけれども，文法的には間違っていない場合に，警告メッセージが出力されます。また，プログラム中にマシンに依存しそうな部分を見つけた場合にも警告メッセージを出力します。

コンパイラは，メッセージの先頭にエラーのクラス（Fatal error，Error，Warning）を表示し，次にエラーを検出したファイルの名前と行番号，最後にメッセージそのものを表示します。

以下では，各クラスごとにアルファベット順にメッセージを示し，考えられる原因と対策もあわせて示します。なお，このリスト内のメッセージの中の'XXXXXXXX'は，実際のメッセージではソースコードで使われている識別名やファイル名に置き換えられます。

エラーメッセージの行番号については注意しなければならないことがあります。Cでは，1行の中に文をどのように書くかについては制限がないので，実質的なエラーの原因は，指摘された行番号より前にある場合がしばしばあります。以下のリストでも，実質的な原因よりも後で検出されがちなメッセージについては，なるべくそれを指摘するようにしています。

致命的なエラー（Fatal）

Bad call of in-line function

マクロ定義からインライン関数を取り出して使用していますが，正しく呼び出していません。インライン関数は，2つの連続した下線（__）で始まりかつ終わっているものです。

Irreducible expression tree

これはコンパイラ自身のエラーを表わすものです。ソースファイルの指摘された行に，コード生成のフェーズで，コードを作り出すことができない式が存在します。その式になんらかの原因が考えられる場合は，その式は取り除くべきです。このエラーが出た場合は，MSA カスタマーサポートセンターに連絡してください。

Register allocation failure

これはコンパイラ自身のエラーを表わすものです。ソースファイルの指摘された行に，コード生成のフェーズで，コードを作り出すことができない式が存在します。その式になんらかの原因が考えられる場合は，式を単純なものにしてください。それでもうまくいかない場合は，その式は取り除くべきです。このエラーが出た場合は，MSA カスタマーサポートセンターに連絡してください。

エラー

 # operator not followed by macro argument name

マクロ定義において，#はマクロ引数を文字列にすることを指示するために使うことができます。#の後にはマクロ引数名が続かなければなりません。

'XXXXXXXX' not an argument

ソースファイルの中で，指摘された識別名が関数の引数として宣言されていますが，関数引数リストにその識別名がありません。

Ambiguous symbol 'XXXXXXXX'

構造体のフィールド名が，複数の構造体の中で，異なるオフセット，あるいは異なる型，またその両方で使われています。そのフィールドを参照するために使われた変数または式が，そのフィールドを含む構造体を示していません。構造体を正しい型にキャストするか，フィールド名が間違っているのであれば正しいものに修正してください。

Argument # missing name

関数の定義に使用された関数プロトタイプにおいて，引数名が省略されています。関数をプロトタイプで定義する場合，プロトタイプには引数名を書かなければなりません。

Argument list syntax error

関数を呼び出す際の引数は，カンマで区切り，最後は右カッコ')'で閉じなければなりません。ソースコード中の引数の後に，カンマあるいは右カッコ以外の文字が続いています。

Array bounds missing]

ソースファイル中で配列が宣言されていますが，配列の範囲が右大カッコ']'で閉じられていません。

Array size too large

宣言された配列が大きすぎて，メモリにおさまりません。

Assembler statement too long

インラインアセンブリ文が480バイトを越えてしまいます。

Bad configuration file

TURBOC.CFGファイルの中に，コメントでないテキストで，コマンドラインオプションとして正しくないものがあります。コマンドラインオプションは，マイナス符号（-）で始まっていなければなりません。

Bad file name format in include directive

インクルードされるファイル名は，二重引用符（"*filename.h*"）または不等号（<*filename.h*>）で囲まれていなければなりません。ファイル名の左側にそのどちらかがつけられていません。マクロが使われている場合は，展開されたテキストが正しくありません。

Bad ifdef directive syntax

#ifdef指令は，指令の本体としては，識別名は1個しか含むことはできません。

Bad ifndef directive syntax

#ifndef指令は，指令の本体としては，識別名は1個しか含むことはできません。

Bad undef directive syntax

#undef指令は，指令の本体としては，識別名は1個しか含むことはできません。

Bit field size syntax

ビットフィールドの幅は，1～16の間の定数で定義しなければなりません。

Call of non-function

関数として宣言されていないものを呼び出しています。これは，関数の宣言が正しくないか，関数名をスペルミスしたことに原因があります。

Cannot modify a const object

constと宣言されたオブジェクトに対して，代入などの許されない操作が行なわれています。

Case outside of switch

switch 文の外側に case 文がありました。おそらく'{'と'}'が正しく対応していません（どちらかがない）。

Case statement missing :

case 文では，定数式の後にコロン（:）が必要です。case 文の式にコロンをつけるのを忘れたか，コロンの前によけいなシンボルがついているはずです。

Cast syntax error

キャストが正しくないシンボルを含んでいます。

Character constant too long

文字定数は，1文字か2文字の長さしか許されません。

Compound statement missing }

ソースファイルの終わりまで達したのに，右カッコ'}'がありませんでした。おそらく'{'と'}'が正しく対応していないことによります。

Conflicting type modifiers

これは，1つのポインタの宣言で，予約語 near と far の両方を指定したときなどに出力されます。1つのポインタには1つのアドレシング修飾子，1つの関数には1つの言語修飾子（cdecl，pascal，または interrupt）しか指定できません。

Constant expression required

配列は定数の大きさで宣言されなければなりません。このエラーは，# define 定数のスペルミスによってよく起こります。

Could not find file 'XXXXXXXX.XXX'

コマンドラインで指定されたファイルが見つかりませんでした。

Declaration missing ;

ソースファイルの中に，最後にセミコロン（;）のない struct または union のフィールド宣言が含まれています。

Declaration needs type or storage class

宣言には，少なくとも型か記憶クラスの指定が必要です。たとえば，次のような宣言文は正しくありません。

```
i,j;
```

Declaration syntax error

ソースファイルの中に，シンボルが欠けている，あるいはよけいなシンボルを含む宣言があります。

Default outside of switch

switch 文の外側に default 文がありました。おそらく'{'と'}'が正しく対応していません。

Define directive needs an identifier

define の後の（ホワイトスペース以外の）最初の文字は，識別名でなければなりません。識別名とは考えられない文字がありました。

Division by zero

ソースファイルの中に，ゼロで割る式があります。

Do statement must have while

do 文の中に，予約語 while がありません。

Do-while statement missing (

do 文の中で，予約語 while の後に左カッコ'('がありません。

Do-while statement missing)

do 文の中で，条件式の右カッコ')'がありません。

Do-while statement missing ;

do 文の中で，条件式の右カッコの後にセミコロン（;）がありません。

Duplicate case

switch 文の各 case の定数式は，他の case と異なるものでなければなりません。

Enum syntax error

enum 宣言の識別名の並びが正しくありません。

Enumeration constant syntax error

enum の値として与えられている式が定数ではありません。

Error Directive： XXXX

このメッセージは，ソースファイル中の#error 指令が実行されたときに表示されます。指令のテキストがメッセージの中に表示されます。

Error writing output file

このエラーは，作業用ディスクがいっぱいになった場合によくでます。ディスケットが壊れているような場合にもでます。いっぱいの場合は，不要なファイルを消去して，もう一度コンパイルを行なってください。

Expression syntax

式の解析において重大なエラーがあった場合に，このメッセージが出力されます。2つの演算子が連続して現われたり，カッコの対応していなかったり，前の文のセミコロンがない場合などです。

Extra parameter in call

プロトタイプで定義されたポインタによる関数呼び出しで，引数が多すぎます。

Extra parameter in call to XXXXXXXX

（プロトタイプで宣言された）関数の呼び出しで，引数が多すぎます。

File name too long

#include 指令で指定されたファイル名が長すぎて，コンパイラが処理できません。MS-DOS におけるファイル名は64文字以下でなければなりません。

For statement missing （

for 文で，for の後に左カッコ'('がありません。

For statement missing ）

for 文で，制御式の後に右カッコ')'がありません。

For statement missing ;

for 文で，式の後にセミコロン';'がありません。

Function call missing)

関数呼び出しの引数の並びに構文エラーがあります。たとえば，右カッコの指定を忘れたりした場合です。

Function definition out of place

関数定義は，他の関数の内部に置くことはできません。関数の内部にある，引数リストをともなう関数の始まりのような宣言は，関数定義とみなされます。

Function doesn't take a variable number of arguments

可変個の引数をとることができない関数で，**va_start** マクロが使用されています。

Goto statement missing label

予約語 goto の後には，識別名が必要です。

If statement missing (

if 文において，予約語 if の後に左カッコ'('がありません。

If statement missing)

if 文において，条件式の後に右カッコ')'がありません。

Illegal character 'C'(0xXX)

入力ファイルの中に無効な文字が使われています。その文字の16進値も表示されます。

Illegal initialization

初期化は，定数式，または，グローバルな外部（extern）変数あるいは static 変数のアドレスに定数を足すか，引くかしたものでなければなりません。

Illegal octal digit

8進定数の中に，8進数字としては許されない数字（8または9）があります。

Illegal pointer subtraction

これは，ポインタでないものからポインタを引こうとしたときに出力されます。

Illegal structure operation

構造体に使用できる演算子は，ピリオド(.)，アドレス of 演算子(&)，代入演算子(=)であり，また構造体は関数への（からの）引数として指定できます。構造体がこれ以外の演算子とともに使用されていると，このエラーが出ます。

Illegal use of floating point

浮動小数点オペランドに対して，シフト，ビット論理演算，条件式（？ ：），間接参照（*）などの演算を行なうことはできません。浮動小数点オペランドが，これらの使用できない演算子とともに使われています。

Illegal use of pointer

ポインタに対しては，加算，減算，代入，関係，間接（*），矢印（->）の演算子しか使用できません。ソースファイルの中で，ポインタが他の演算子とともに使われています。

Improper use of a typedef symbol

式の中で，typedef 名が，変数を置くべき場所に使われています。typedef 宣言とスペルミスがないかどうかチェックしてください。

In-line assembly not allowed

ソースファイルにはインラインアセンブリ文が含まれていますが，それを統合環境の中からコンパイルしようとしています。このソースファイルをコンパイルするには，TCC コマンドを使用する必要があります。

Incompatible storage class

ソースファイルの中で，関数定義に対して予約語 extern が使われています。関数定義には，記憶クラスとしては，何もなしか，つけるとすれば static しか使用できません。

Incompatible type conversion

ある型から別の型への変換をしようとしていますが，この2つの型はコンパチブルではありません。これには，関数と非関数の間，構造体や配列とスカラ型の間，浮動小数点値とポインタ型の間などの変換が含まれます。

Incorrect command line argument： XXXXXXXX

このコマンドライン引数は，正当なものとは認められません。

Incorrect confiugration file argument : XXXXXXXX

コンフィギュレーションファイル内のこの引数は，正当なものとは認められません。特に，マイナス符号（-）がついているかチェックしてください。

Incorrect number format

16進表記の中に小数点が現われました。

Incorrect use of default

予約語 default の後にコロン（：）がありません。

Initializer syntax error

初期化において，演算子がたりない，余計な演算子がある，カッコが正しく対応していない，など書式として受け入れられません。

Invalid indirection

間接演算子（*）のオペランドは，void ではないポインタでなければなりません。

Invalid macro argument separator

マクロ定義内の引数は，カンマで区切られていなければなりません。引数名の後に別の文字がありました。

Invalid pointer addition

2つのポインタを足し合わせようとしています。

Invaild use of arrow

矢印演算子（->）の後ろには，識別名を置かなければなりません。

Invaild use of dot

ドット演算子（.）の後ろには，識別名を置かなければなりません。

Lvalue required

代入演算子の左辺は，値を代入するアドレスが特定できる式でなければなりません。これには，数値変数，ポインタ変数，構造体フィールド参照，ポインタによる間接参照，配列要素などが含まれます。

Macro argument syntax error

マクロ定義における引数は識別名でなければなりません。引数が現われるべき位置に識別名には許されない文字が使われています。

Macro expansion too long

マクロは4096文字以上に展開することはできません。このエラーは，マクロがそれ自身を再帰的に展開しようとする場合に起こります。マクロは自分自身を正しく展開することはできません。

May compile only one file when an output file name is given

コマンドラインオプション-o を指定すると出力ファイルは1個のみとなるので，先頭のファイルだけがコンパイルされ，他のファイルはコンパイルされません。

Mismatched number of parameters in definition

関数定義における引数が，関数プロトタイプで与えられた情報と一致しません。

Misplaced break

switch 文の外側，あるいはループの外側に break 文があります。

Misplaced continue

switch 文の外側，あるいはループの外側に continue 文があります。

Misplaced decimal point

浮動小数点定数の指数部に小数点があります。

Misplaced else

if 文と対応しない else 文があります。これは，単によけいな else がある場合の他に，よけいなセミコロンがある，'{'や'}'を入れ忘れている，前の if 文の中に構文エラーがある場合などに起こります。

Misplaced elif directive

if，# ifdef，# ifndef に対応しない# elif 指令があります。

Misplaced else directive

#if, #ifdef, #ifndef に対応しない#else 指令があります。

Misplaced endif directive

#if, #ifdef, #ifndef に対応しない#endif 指令があります。

Must be addressable

&演算子が，特定のアドレスに置かれないオブジェクト，たとえばレジスタ変数などに使用されています。

Must take address of memory location

アドレス of 演算子（&）が式の中で，許されない使い方をされています。たとえば，レジスタ変数などに対して使用されています。

No file name ending

#include 文のファイル名を閉じる不等号（>）あるいは二重引用符（"）がありません。

No file name given

Turbo C のコンパイラ（TCC）に，ファイル名が指定されていません。ソースファイル名を指定しなければなりません。

Non-portable pointer assignment

ソースファイルの中に，ポインタをポインタでないものへ代入する文，あるいはその逆の代入を行なう文があります。特殊なケースとして，定数0をポインタに代入することは許されています。代入そのものが適切なのであれば，このエラーメッセージが出ないようにするために，キャストを使用してください。

Non-portable pointer comparison

ソースファイルの中で，ポインタとポインタ以外のものとの比較が行なわれています。ポインタは，ポインタ以外のものでは定数0としか比較できません。比較そのものが適切なのであれば，このエラーメッセージが出ないようにするために，キャストを使用してください。

Non-portable return type conversion

return 文の中の式の型が，関数宣言の型と一致していません。1つの例外を除いて，関数または return 文の式がポインタの場合にはこのエラーがでます。例外は，ポインタを返す関数が定数0を返す場合です。0は適切なポインタ値に変換されます。

Not an allowed type

ソースファイルの中で，許されない型，たとえば関数を返す関数，配列を返す関数などを宣言しています。

Out of memory

使用できるメモリがなくなりました。メモリを増設する必要があります。すでにメインメモリに640KB を実装しているのであれば，デバイスドライバを減らして利用できるメモリをふやすか，プログラムを分割するなどしてソースファイルを小さくすることを考えてください。

Pointer required on left side of ->

矢印演算子（->）の左側にはポインタしか置けません。

Redeclaration of 'XXXXXXXX'

この識別名は，以前に宣言されています。

Size of structure or array not known

sizeof のような式が，未定義の構造体や長さが空の配列とともに使われました。構造体は，その大きさが必要なければ，定義の前に参照することができます。配列は，メモリ領域を確保するのではない場合や，長さを与える初期化子が後に続く場合は，長さを空にして宣言ができます。

Statement missing ;

最後にセミコロンがついていない文があります。

Structure or union syntax error

予約語 struct または union の後に，識別名あるいは '{'がありません。

Structure size too large

メモリに入りきらない大きさの構造体を宣言しています。

Subscripting missing]

右カッコ']'がない添字式があります。これは演算子を入れなかったり，余計に入れてしまったり，カッコの対応が悪い時にでるエラーです。

Switch statement missing (

switch 文で，キーワード switch の後に左カッコ'('がありません。

Switch statement missing)

switch 文で，判定式の後に右カッコ')'がありません。

Too few paraments in call

（関数ポインタによる）プロトタイプを持つ関数への呼び出しにおいて，引数の数が少なすぎます。プロトタイプは，すべての引数が渡されることを要求します。

Too few parameters in call to 'XXXXXXXX'

（プロトタイプで宣言された）関数の呼び出しにおいて，引数の数がたりません。

Too many cases

switch 文のケースの数の上限は257です。

Too many decimal points

浮動小数点定数中に小数点が複数ありました。

Too many default cases

1つの switch 文の中に default 文が複数ありました。

Too many exponents

浮動小数点定数の中に指数部が複数ありました。

Too many initializers

宣言によって許される個数以上の初期化子が存在します。

Too many storage classes in declaration

1つの宣言は，複数個の記憶クラスを持つことはできません。

Too many types in declaration

1つの宣言は，複数個の型を持つことはできません。型は次の基本型のいずれかです。

char, int, float, double, struct, union, enum, typedef 名

Too much auto memory in function

現在の関数は，使用できるメモリ以上の自動記憶領域を宣言している。

Too much code defined in file

現在のソースファイルの中の関数を合わせた大きさが64Kバイトを越えました。必要のないコードを削るか，ソースファイルを分割するかしてください。

Too much global data defined in file

グローバルなデータ宣言の総計が64Kバイトを越えました。大きすぎる配列がないか調べてください。全部が必要な場合は，プログラムの再編成を考えてください。

Two consecutive dots

3つの連続したピリオド（...）は省略を意味し，1個のピリオドは小数点やメンバを指定するのに使われます。しかし，2つの連続するピリオドはCでは使われません。

Type mismatch in parameter #

（関数ポインタによって呼び出される）関数がプロトタイプとともに宣言されていて，左から#N番目の引数は宣言の引数型に変換されることができませんでした。

Type mismatch in parameter # in call to 'XXXXXXXX'

ソースファイルの中で，関数がプロトタイプとともに宣言されていますが，左から#N番目の引数を，宣言の引数型に変換することができませんでした。

Type mismatch in parameter 'XXXXXXXX'

ソースファイルの中で，関数ポインタによって呼び出される関数がプロトタイプとともに宣言されていますが，引数を宣言の引数型に変換することができませんでした。

Type mismatch in parameter 'XXXXXXXX' in call to 'YYYYYYYY'

ソースファイルの中で，関数がプロトタイプとともに宣言されていますが，引数を宣言の引数型に変換することができませんでした。

Type mismatch in redeclaration of 'XXX'

ソースファイルの中で，ある変数が，もともと宣言された型とは異なる型に再宣言されています。これは，ある関数がまず呼び出され，その後で整数とは違うものを返すと宣言された場合に起こります。このエラーが出る場合は，その関数が最初に呼び出される前に，関数の extern 宣言を入れます。

Unable to create output file 'XXXXXXXX.XXX'

このエラーは，作業用ディスクがいっぱいになったり，ライトプロテクトされている場合に出ます。いっぱいの場合は，不必要なファイルを削除し，コンパイルをやり直します。ライトプロテクトされている場合は，そのソースファイルを書き込み可能なディスクにコピーして，コンパイルをやり直してください。

Unable to create turboc.lnk

コンパイラがテンポラリファイル TURBO.$LN を作成することができません。このエラーは，コンパイラがディスクをアクセスできないか，ディスクがいっぱいの場合に出ます。

Unable to execute command 'XXXXXXXX'

TLINK または TASM が見つかりません。またはディスクが壊れています。

Unable to open include file 'XXXXXXXX.XXX'

コンパイラが指定のファイルを見つけられませんでした。このエラーは，インクルードファイルが自分自身をインクルードしている場合にも出ます。指定のファイルが存在しているかチェックしてください。

Unable to open input file 'XXXXXXXX.XXX'

このエラーは，ソースファイルが見つからなかった場合に出ます。ファイル名のスペルと，そのファイルが指定のディスクやディレクトリに存在しているかどうかをチェックしてください。

Undefined label 'XXXXXXXX'

goto 文で使用されているラベルの定義がありません。

Undefined structure 'XXXXXXXX'

構造体を，エラーが示されているところより前で使用していますが，その構造体に対する定義がありません。このエラーは，構造体名をスペルミスしたか，宣言を忘れた場合にでます。

Undefined symbol 'XXXXXXXX'

この識別名は宣言されていません。これは，この行あるいは宣言を行なっている行でスペルミスをしていることによって起こります。この識別名の宣言でエラーが起こっている場合にも，このエラーはでます。

Unexpected end of file in comment started on line #

コメントの途中で，ソースファイルが終わっています。このエラーは，通常はコメントの終わりの記号（*/）を忘れたことで起こります。

Unexpected end of file in conditional started on line #

コンパイラが#endifに出会う前にソースの終わりが現われました。#endifを入れ忘れたか，スペルミスが原因です。

Unknown preprocessor directive： 'XXX'

行の初めに文字#がありますが，それに続く指令名が次のいずれでもありませんでした。

define, undef, line, if, ifdef, ifndef, include, else, endif

Unterminated character constant

対応していない単引用符があります。

Unterminated string

対応していない二重引用符があります。

Unterminated string or character constant

文字列あるいは文字定数が始まっていますが，それを終了させる引用符がありません。

User break

統合環境でコンパイル中あるいはリンク中に，***CTRL-STOP***（***Ctrl-Break***）が入力されました。

While statement missing (

while 文で，判定式の後の右カッコ'('がありません。

While statement missing)

while 文で，判定式の後の右カッコ')'がありません。

Wrong number of arguments in call of 'XXXXXXXX'

マクロの呼び出しで，引数の個数が定義と一致していません。

警告

'XXXXXXXX' declared but never used

指定の変数がソースファイル中で宣言されていますが，全く使われていません。この警告は，複合文，つまり関数の終わりのカッコ'}'にコンパイラが出会った段階で出力されます。変数の宣言は複合文，つまり関数の初めで始まります。

'XXXXXXXX' is assigned a value which is never used

変数は代入文には使われていますが，その他では全く使われていません。この警告は，コンパイラが終わりの'}'カッコに出会った段階で出力されます。

'XXXXXXXX' not part of structure

指定のフィールドは，ドット（.）または矢印（->）の左側にある構造体の一部分でないか，あるいは左側にあるものが構造体でない（ドットの場合）か，構造体を指すポインタ（矢印の場合）ではないことを意味します。

Ambiguous operators need parentheses

この警告は，2つのシフト演算子，関係演算子，ビット論理演算子がカッコなしに連続して現われたときに出力されます。また，加算あるいは減算演算子がカッコなしにシフト演算子とともに現われた場合にもこの警告が出ます。演算子の優先順位は，やや直感的にとらえにくい点があり，プログラマはよく優先順位を間違えがちです。

Both return and return of a value used

この警告は，コンパイラが関数の中の前の return 文とは一致しない return 文に出会った段階で出力されます。値を返す return 文と，値を返さない return 文があるということなので，これはおそらくエラーです。

Call to function with no prototype

このメッセージは，"Prototypes required"警告が許可されて，プロトタイプを最初に与えずにその関数を呼び出した場合に出力されます。

Call to function 'XXXX' with no prototype

このメッセージは，"Prototypes required"警告が許可されていて，プロトタイプを最初に与えずに関数 XXXX を呼び出した場合に出力されます。

Code has no effect

この警告は，コンパイラが，何の効果もない演算子を含む文があったときに出力されます。たとえば次の文は，

```
a + b;
```

どちらの変数に対しても何も影響を与えません。こうしたものは不要であり，なんらかのバグを意味していることもあります。

Constant is long

コンパイラが，32767より大きい10進定数，あるいは65535より大きい8進（または16進）定数で最初に l や L の文字がついていないものがあった場合に出力されます。これらの定数は long として扱われます。

Constant out of range in comparison

ソースファイル中に，ある定数部分式がもう1方の部分式の型によって許される範囲の外側にある場合の比較が含まれています。たとえば unsigned の値を -1と比較するような場合です。32767（10進）より大きい unsigned 型にキャストする（たとえば(unsigned) 65535）か，定数の最後に u または U をつけます（たとえば65535u)。
このメッセージが出力されても，コンパイラは比較のためのコードを生成します。このコードが常に同じ結果を与えるもの（文字式を4000と比較するといったもの）であっても，必らず判定は行なわれます。これにより unsigned 型の式と -1を比較しても意味があるということになります。実際（unsigned) 65535と -1は，8086上では同じビットパターンを与えるからです。

Conversion may lose significant digits

代入文などにおいて，long あるいは unsigned long から，int あるいは unsigned int への変換が行なわれる場合があります。マシンによっては，int と long の変数が占めるバイト数が同じ場合もあるので，この種の変換がプログラムの動作を変えてしまうことがあります。

このメッセージを出力した場合には，コンパイラは比較を行なうコードを生成します。char 型の式を4000と比較するような場合に，生成したコードが常に同じ結果を示すとしても，このコードはやはり比較を行ないます。これは，unsigned の式と-1との比較でも有効な場合があることを意味します。8086においては，unsigned の値が，-1と同じビットパターンをとることもあり得るからです。

Function should return a value

この関数は，int 型や void 型以外の型の値を返すと宣言されているのに，値を返さない return 文が使われています。おそらくはエラーです。古いタイプのCでは，値を返さない関数であることを意味する void 型がなかったので，int 型と宣言されている場合はこのメッセージは出ません。

Hexadecimal or octal constant too large

文字列リテラルあるいは文字定数において，255を越える値（たとえば¥777や¥x1234）を持つ16進または8進エスケープシーケンスが指定されています。

Mixing pointers to signed and unsigned char

char ポインタから unsigned char ポインタへの変換（あるいはその逆）を，キャストなしで行なっています（厳密にいうとこれは間違いですが，8086ではあまり問題になりません）。

No declaration for function 'XXXXXXXX'

これは，この警告がオンになっていて(-wnod)，関数を宣言せずに呼び出している場合に出力されます。宣言は古典・モダン（プロトタイプ），いずれのスタイルでもかまいません。

Non-portable pointer assignment

ポインタをポインタでないものへ代入（あるいはその逆）しています。ポインタに0を代入することは，特別な場合として許されています。この警告を出さないようにキャストすべきです。

Non-portable pointer comparison

ポインタを0以外のポインタでない値と比較しています。比較が適切な場合，この警告を出さないようキャストすべきです。

Non-portable return type conversion

return 文の中の式の型が，関数宣言の型と同じではありません。関数または戻り値がポインタの場合にこのエラーがでます。唯一の例外は，ポインタを返す関数が定数0を返す場合です。0は適当なポインタ値に変換されます。

Parameter 'XXXXXXXX' is never used

関数の中で宣言された引数が，関数本体の中で使用されていません。エラーかどうかは確定できませんが，引数のスペルミスによってよく起こります。また関数の本体で自動（ローカル）変数として再宣言されている場合もあります。この場合，引数が自動変数によってマスクされ，未使用のままになります。

Possible use of 'XXXXXXXX' before definision

変数に値が代入される前に，その変数が式の中で使われています。この点に関してコンパイラが行なう検査は簡単なもので，変数の使用が代入よりもソースコード中で物理的に前にあると，この警告が出ます。実際のプログラムの流れでは，使用する前に代入されるのかもしれません。

Possibly incorrect assignment

この警告は，if 文，while 文，do-while 文などで，条件式の主たる演算子が代入演算子であった場合に出力されます。==演算子とまちがえて，=を使っている場合によく起こるものです。この警告を抑制するには，代入文をカッコで囲み，その全体としての値とゼロとの比較を明示的に行なってください。たとえば次の式を，

```
if (a = b) ...
```

次のように書き変えます。

```
if ((a = b) != 0) ...
```

Redefinition of 'XXXXXXXX' is not identical

前に定義されているマクロを，別のテキストで再定義しています。この場合マクロは新しいテキストで置き換えられます。

Restarting compile using assembly

-B コンパイラオプション，あるいは# pragma inline 文の指定なしで，asm 文に出会いました。アセンブリ言語機能を使用して，コンパイルをリスタートします。

Structure passed by value

この警告をオンにしている場合（-wstv），構造体が引数として値で渡されていると，この警告がでます。よくあるのは，引数として構造体のアドレスを渡すべきところで，&演算子をつけ忘れることです。構造体は値で渡すこともできるのでエラーとはなりません。この警告は，プログラムミスを気づかせるためのものです。

Superfluous & with function or array

アドレス of 演算子（&）は配列名や関数名につける必要はありません。この場合&演算子は削除されます。

Suspicious pointer conversion

異なる型を指すポインタの間で変換を行なっています。その変換が適切なものであるならば，この警告を出さないようにするためにはキャストを使用してください。

Undefined structure 'XXXXXXXX'

ソースファイルの中で使用されている構造体 XXXXXXXX は，定義されていません。これは，構造体名のスペルミス，あるいは宣言がないことによって起こります。

Unknown assembler instruction

インラインアセンブリ文に許されない命令コードが使われています。命令コードをチェックしてください。命令が受け入れられるものかどうか，許されている命令コードの一覧表もチェックしてください（ユーザーズガイド第12章）。

Unreachable code

break，continue，goto，return の後に，ラベル，あるいはループや関数の終わりが続いていません。コンパイラは，条件判定式が定数の while，do，for ループをチェックし，無限ループを認識しようとします。

Void functions may not return a value

この関数は void 型と宣言されているのに，値を返そうとする return 文があります。return 文の値は無視されます。

Zero length structure

大きさがゼロの構造体が宣言されています。この構造体を使用するとエラーになります。

付録 B

言語の構文の要約

ここでは，Turbo C における構文の要約を行ないますが，BNF(バッカス-ナウア記法)を少し修正したものを使用することにします。次のように3つのカテゴリにわけて説明を行ないます。

- **■語意に関する文法**：トークン，キーワード，識別名，定数，文字列リテラル，演算子，区切子
- **■フレーズの構文に関する方法**：式，宣言，文，外部定義
- **■プリプロセッサ指令**

構文中のオプション（省略可能）の要素は，不等号記号（〈...〉）で囲んであります。

語意に関する文法

トークン

トークン：

キーワード

識別名

定数

文字列

演算子

区切子

キーワード（予約語）

キーワード：以下のうちの1つ

asm	do	goto	return	union
auto	double	huge	short	unsigned
break	else	if	signed	void
case	enum	int	sizeof	volatile
cdecl	extern	interrupt	static	while
char	far	long	struct	_cs
const	float	near	switch	_ds
continue	for	pascal	typedef	_es
default		register		_ss

識別名

識別名：

非数字

識別名　非数字

識別名　数字

非数字：以下のうちの1つ

a b c d e f g h i j k l m n o p q r s t u v w x y z _ $

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

数字：以下のうちの1つ

0 1 2 3 4 5 6 7 8 9

定数

定数：

浮動小数点小数

整定数

列挙型定数

文字定数

浮動小数点定数：

小数点定数　〈指数部〉〈浮動小数点接尾辞〉

数字の並び　指数部　〈浮動小数点接尾辞〉

小数点定数：

〈数字の並び〉. 数字の並び

数字の並び .

指数部：

e 〈符号〉 数字の並び

E 〈符号〉 数字の並び

浮動小数点接尾辞

\+ -

数字の並び：

数字

数字の並び 数字

浮動小数点接尾辞：以下のうちの1つ

f l F L

整定数：

10進定数 〈整数接尾辞〉

8進定数 〈整数接尾辞〉

16進定数 〈整数接尾辞〉

10進定数：

非ゼロ数字

10進定数 数字

8進定数：

0

8進定数 8進数字

16進定数：

0x16進数字

0X16進数字

16進定数 16進数字

非ゼロ数字：以下のうちの1つ

1 2 3 4 5 6 7 8 9

8進数字：以下のうちの1つ

0　1　2　3　4　5　6　7

16進数字：以下のうちの1つ

0　1　2　3　4　5　6　7　8　9

a　b　c　d　e　f

A　B　C　D　E　F

整数接尾辞：

符号なし接尾辞　〈倍長接尾辞〉

倍長接尾辞　〈符号なし接尾辞〉

符号なし接尾辞：以下のうちの1つ

u　U

倍長接尾辞：以下のうちの1つ

l　L

列挙型定数：

識別子

文字定数：

c **文字の並び**

c **文字の並び**：

c **文字**

c **文字の並び**　*c* **文字**

c **文字**：

ソース記述文字セットの文字ならどんな文字でも。ただし，単引用符（'），円記号（¥），改行文字を除く。

エスケープシーケンス

エスケープシーケンス：以下のうちの1つ

¥'	¥b	¥v	¥xhh
¥"	¥f	¥o	¥xhhh
¥?	¥n	¥oo	¥Xh
¥¥	¥r	¥ooo	¥Xhh
¥a	¥t	¥xh	¥Xhhh

文字列リテラル

文字列リテラル：

"〈*s* **文字の並び**〉"

s **文字の並び**：

s **文字**

s **文字の並び**　*s* **文字**

s **文字**：

ソース記述文字セットの文字ならどんな文字でも。ただし，二重引用符（"），円記号（¥），改行文字を除きます。

エスケープシーケンス

演算子

演算子：以下のうちの1つ

[]　()　.　->　++　--

&　*　+　-　~　!

sizeof　/　%　<<　>>　<

>　<=　>=　==　!　=

^　|　&&　| |　?：　=

*=　/=　%=　+=　-=　<<=

>>=　&=　=　=　.　#

##

区切子

区切子：以下のうちの1つ

[]　()　{}　*　,　:

=　;　...　#

フレーズの構造に関する文法

式

一次式：

　識別子

　定数

　疑似変数

　文字列リテラル

　(式)

疑似変数

_AX	_AL	_AH	_SI	_ES
_BX	_BL	_BH	_DI	_SS
_CX	_CL	_CH	_BP	_CS
_DX	_DL	_DH	_SP	_DS

ポストフィックス式：

　一次式

　ポストフィックス式 [式]

　ポストフィックス式 (<引数式リスト>)

　ポストフィックス式 . 識別子

　ポストフィックス式 -> 識別子

　ポストフィックス式 ++

　ポストフィックス式 --

引数式リスト：

　代入式

　引数式リスト , 代入式

単項式：

ポストフィックス式

＋＋単項式

--単項式

単項演算子　キャスト式

sizeof　単項式

sizeof　（ 型名 ）

単項演算子：以下のうちの1つ

&　　*　　+　　-　　～　　!

キャスト式：

単行式

（ 型名 ）キャスト式

乗法式：

キャスト式

乗法式　*　キャスト式

乗法式　/　キャスト式

乗法式　%　キャスト式

加法式：

乗法式

加法式　＋　乗法式

加法式　-　乗法式

シフト式：

加法式

シフト式　<<　加法式

シフト式　>>　加法式

関係式：

シフト式

関係式 <　シフト式

関係式 >　シフト式

関係式 <= シフト式
関係式 >= シフト式

等号式：
関係式
等号式 == 関係式
等号式 != シフト式

AND 式：
等号式
AND 式 & 等号式

排他的 OR 式：
AND 式
排他的 OR 式 ^ AND 式

OR 式：
排他的 OR 式
OR 式 排他的 OR 式

論理 AND 式：
OR 式
論理 AND 式 && OR 式

論理 OR 式：
論理 AND 式
論理 OR 式 論理 AND 式

条件式：
論理 OR 式
論理 OR 式 ? 式 : 条件式

代入式：
条件式
単行式 代入演算子 代入式

代入演算子：以下のうちの1つ

　　=　　*=　　/=　　%=　　+=　　-=

　　<<=　　>>=　　&=　　=　　=

式：

　　代入式

　　式，代入式

定数式：

　　条件式

宣言

宣言：

　　宣言指定子　〈初期値宣言リスト〉

宣言指定子：

　　記憶クラス指定子　〈宣言指定子〉

　　型指定子　〈宣言指定子〉

初期値宣言リスト：

　　初期値宣言子

　　初期値宣言リスト，初期値宣言子

初期値宣言子：

　　宣言子

　　宣言子　=　初期化子

記憶クラス指定子：

typedef
extern
static
auto
register

型指定子：

void
char
short
int
long
float
double
signed
unsigned
const
volatile
構造体・共用体指定子
列挙型指定子
typedef 名

構造体・共用体指定子：

構造体・共用体　〈識別名〉　{ 構造体宣言リスト }
構造体・共用体　識別名

構造体・共用体：

struct
union

構造体宣言リスト：

構造体宣言
構造体宣言リスト　構造体宣言

構造体宣言：

　　型指定子リスト　構造体宣言子リスト；

型指定子リスト：

　　型指定子

　　型指定子リスト　，　型指定子

構造体宣言リスト：

　　構造体宣言子

　　構造体宣言子リスト　，　構造体宣言子

構造体宣言子：

　　宣言子

　　〈宣言子〉　：　定数式

列挙型指定子：

　　enum　〈識別名〉　｛ 列挙子リスト ｝

　　enum　識別名

列挙子リスト：

　　列挙子

　　列挙子リスト　，　列挙子

列挙子：

　　列挙型定数

　　列挙型定数 ＝ 定数式

宣言子：

　　〈ポインタ〉　直接宣言子

　　〈修飾子リスト〉

直接宣言子：

識別名

（ 宣言子 ）

直接宣言子 ［〈定数式〉］

直接宣言子 （ パラメータ型リスト ）

直接宣言子 （〈識別名リスト〉）

ポインタ：

＊〈型指定子リスト〉

＊〈型指定子リスト〉 ポインタ

修飾子リスト：

修飾子

修飾子リスト 修飾子

修飾子：

cdecl

pascal

interrupt

near

far

huge

パラメータ型リスト：

パラメータリスト

パラメータリスト ,...

引数リスト：

引数宣言

引数リスト ， 引数宣言

パラメータ宣言：

宣言指定子 宣言子

宣言指定子 〈抽象宣言子〉

識別名リスト：

識別名

識別名リスト ， 識別名

型名：

型指定リスト 〈抽象宣言子〉

抽象宣言子：

ポインタ

〈ポインタ〉 〈直接抽象宣言子〉

〈修飾子リスト〉

直接抽象宣言子：

(抽象宣言子)

〈直接抽象宣言子〉 [〈定数式〉]

〈直接抽象宣言子〉 (〈パラメータ型リスト〉)

typedef 名：

識別名

初期化子：

代入式

{ 初期化子リスト }

{ 初期化子リスト,}

初期化子リスト：

初期化子

初期化子リスト, 初期化子

文

文：

　ラベルつき文
　複合文
　式文
　選択文
　繰り返し文
　飛び越し文
　asm 文

asm 文：

　asm　トークン　改行
　asm　トークン　；

ラベルつき文：

　識別名　：　文
　case　定数式　：　文
　default　：　文

複合文：

　{〈宣言リスト〉　〈文リスト〉}

宣言リスト：

　宣言
　宣言リスト　宣言

文リスト：

　文
　文リスト　文

式文：

　〈式〉；

選択文：

if（ 式 ）文

if（ 式 ）文　else　文

switch　（ 式 ） 文

繰り返し文：

while（ 式 ）文

do　文　while（ 式 ）；

for　（〈式〉；〈式〉；〈式〉）　文

飛び越し文：

goto　識別名　；

continue　；

break　；

return〈式〉；

外部定義

ファイル：

外部定義

ファイル　外部定義

外部定義：

関数定義

宣言

関数定義：

〈宣言指定子〉　宣言子　〈宣言リスト〉　複合文

プリプロセッサ指令

プリプロセッシングファイル：
　グループ

グループ：
　グループ部
　グループ　グループ部

グループ部：
　〈*pp* トークン〉　改行
　if セクション
　制御行

if セクション：
　if グループ　〈*elif* グループ群〉　〈*else* グループ〉　*endif* 行

if グループ：
　# *if*　定数式　改行　〈グループ〉
　# *ifdef*　識別名　改行　〈グループ〉
　# *ifndef*　識別名　改行　〈グループ〉

elif グループ群：
　elif グループ
　elif グループ群　*elif* グループ

elif グループ：
　# elif　定数式　改行　〈グループ〉

else グループ：
　# else　改行　〈グループ〉

endif 行：

    # endif　改行

制御行

# include	*pp* トークン　改行
# define	識別名　置換リスト　改行
# define	識別名　左カッコ　〈識別名リスト〉　)　置換リスト　改行
# undef	識別名　改行
# line	*pp* トークン　改行
# error	〈*pp* トークン〉　改行
# pragma	〈*pp* トークン〉　改行
# pragma	warn　アクション　短縮語　改行
# pragma	inline　改行
#	改行

アクション：

    +

    -

    .

短縮語：

amb	def	rch	stu
amp	dup	ret	stv
apt	eff	rng	sus
aus	mod	rpt	ucp
big	par	rvl	use
cln	pia	sig	voi
cpt	pro	str	zst

左カッコ：

    前にホワイトスペースがついていない左カッコ

置換リスト：

    〈*pp* トークン群〉

pp **トークン群**：

　　プリプロセッシングトークン

　　pp **トークン群　プリプロセッシングトークン**

プリプロセッシングトークン：

　　ヘッダ名　（#include 指令においてのみ）

　　識別名　（キーワードと重複しないもの）

　　定数

　　文字列リテラル

　　演算子

　　区切子

　　各々は前にホワイトスペースがついていないもの

ヘッダ名：

　　<*h* **文字の並び**>

h **文字の並び**：

　　h **文字**

　　h **文字の並び**　**h 文字**

h **文字**：

　　ソース記述文字セット中のすべての文字。ただし，改行文字と不等号（>）は除く。

改行：

　　改行文字

【A】

【B】

【C】

【D】

【E】

【F】

【G】

【H】

【I】

【J】

【K】

【L】

【M】

【N】

【O】

【P】

【Q】

【R】

【S】

【T】

【U】

【V】

【W】

【X】

【か】

【さ】

【た】

【な】

【は】

【ま】

【や】

【ら】

【わ】